$$\delta : Q \times 2^Q \to Q$$

Distributed Automata and Logic

Fabian Reiter

12 December 2017

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

université
PARIS
DIDEROT
PARIS 7

USPC
Université Sorbonne
Paris Cité

Ultimate objective

Descriptive complexity theory

Distributed computing

Fagin's theorem (1974)

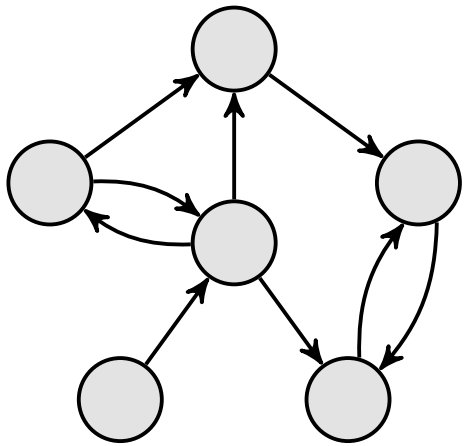
Fagin's theorem (1974)

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

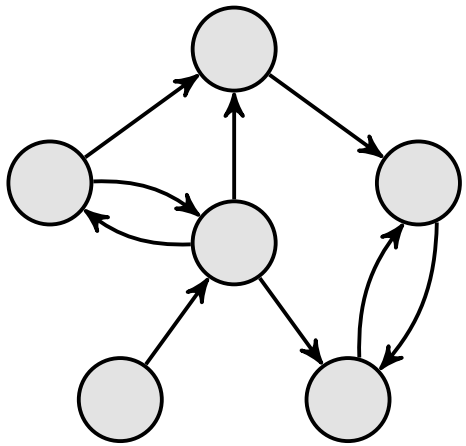
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



Fagin's theorem (1974)

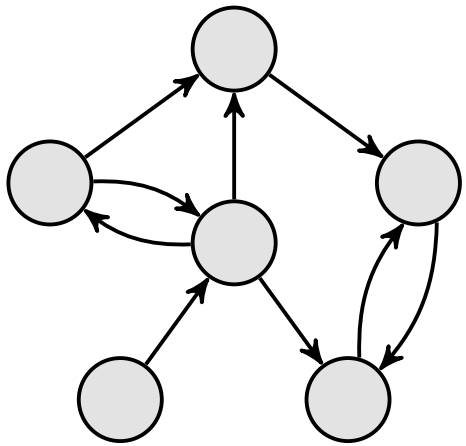
∃ SECOND-ORDER LOGIC



Example: Hamiltonian path

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



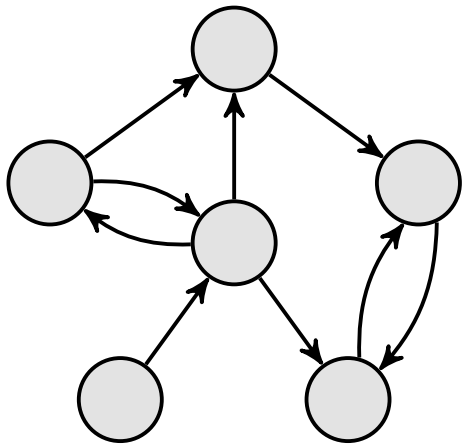
Example: Hamiltonian path

$\exists R \, ($

$)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



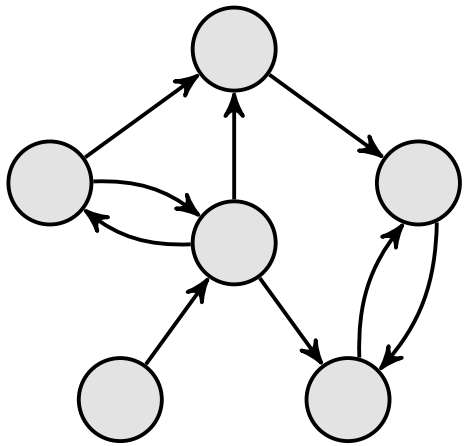
Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$

$\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

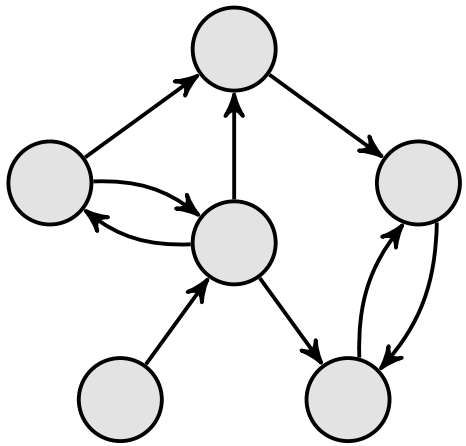


Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



Example: Hamiltonian path

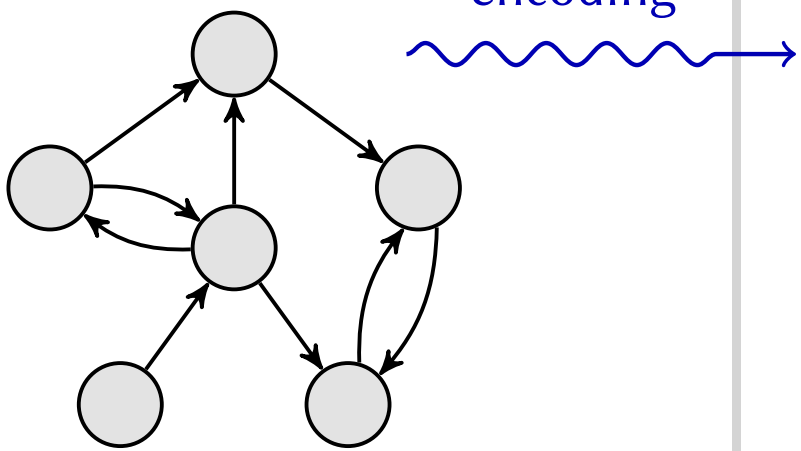
$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

NP TURING MACHINES

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES

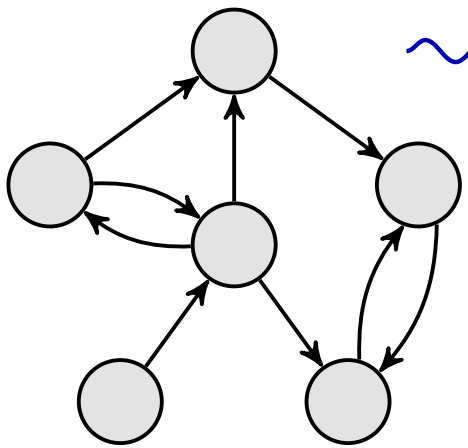


Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

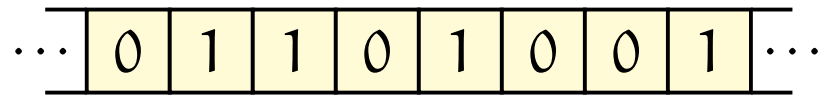
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



encoding

NP TURING MACHINES



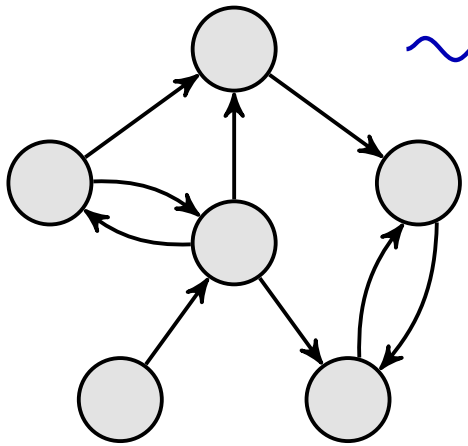
Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

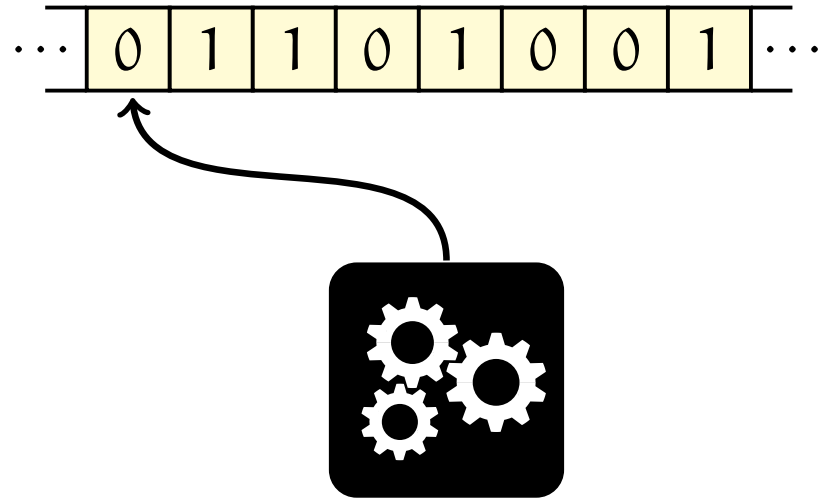
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES



encoding

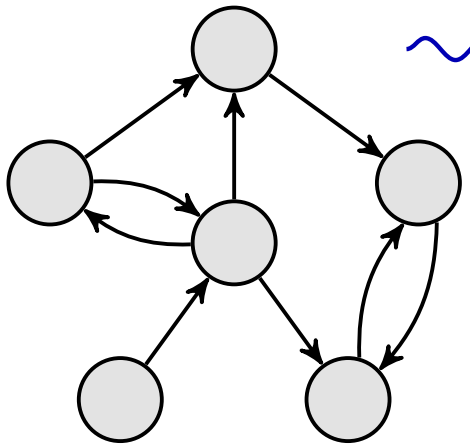


Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

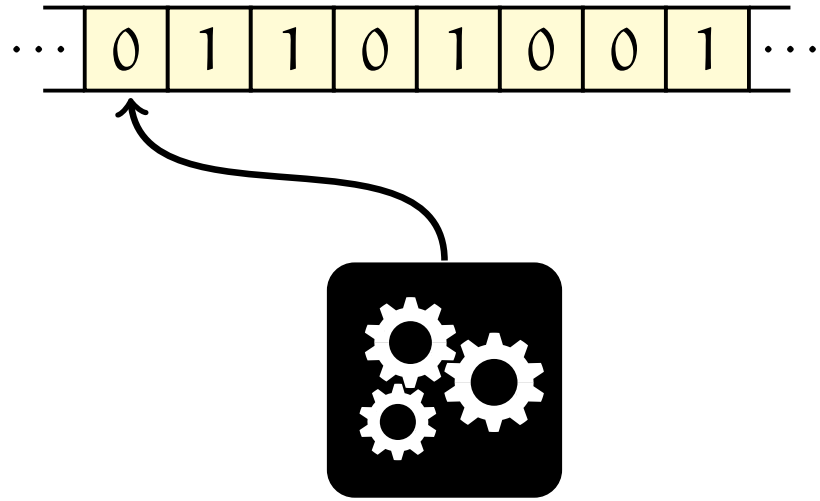


Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

NP TURING MACHINES

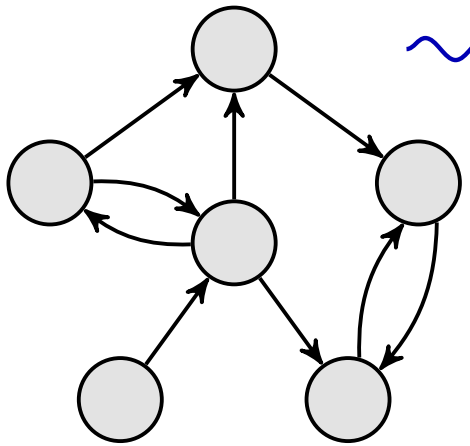
encoding



- Nondeterministic moves

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

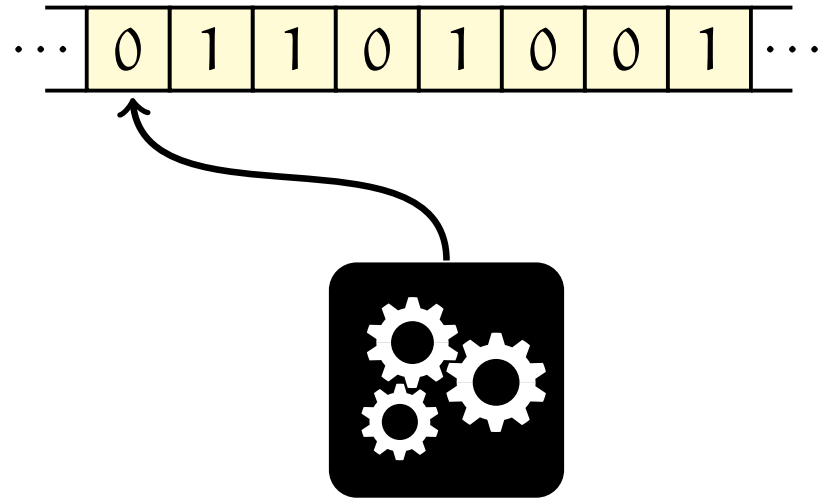


Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

encoding

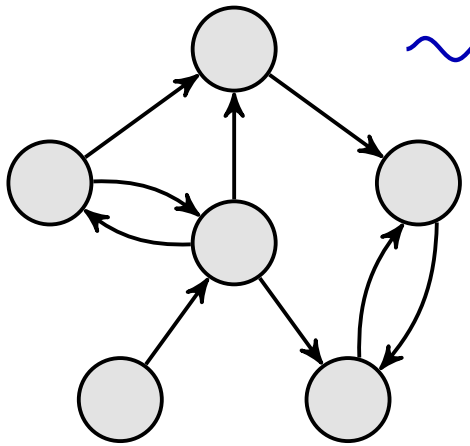
NP TURING MACHINES



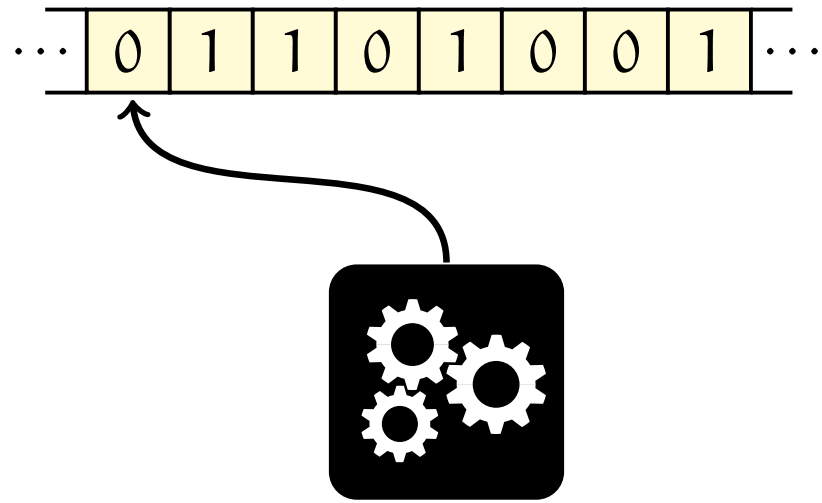
- Nondeterministic moves
- Polynomial running time

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC ⟺ EQUIVALENT ⟺ NP TURING MACHINES



encoding

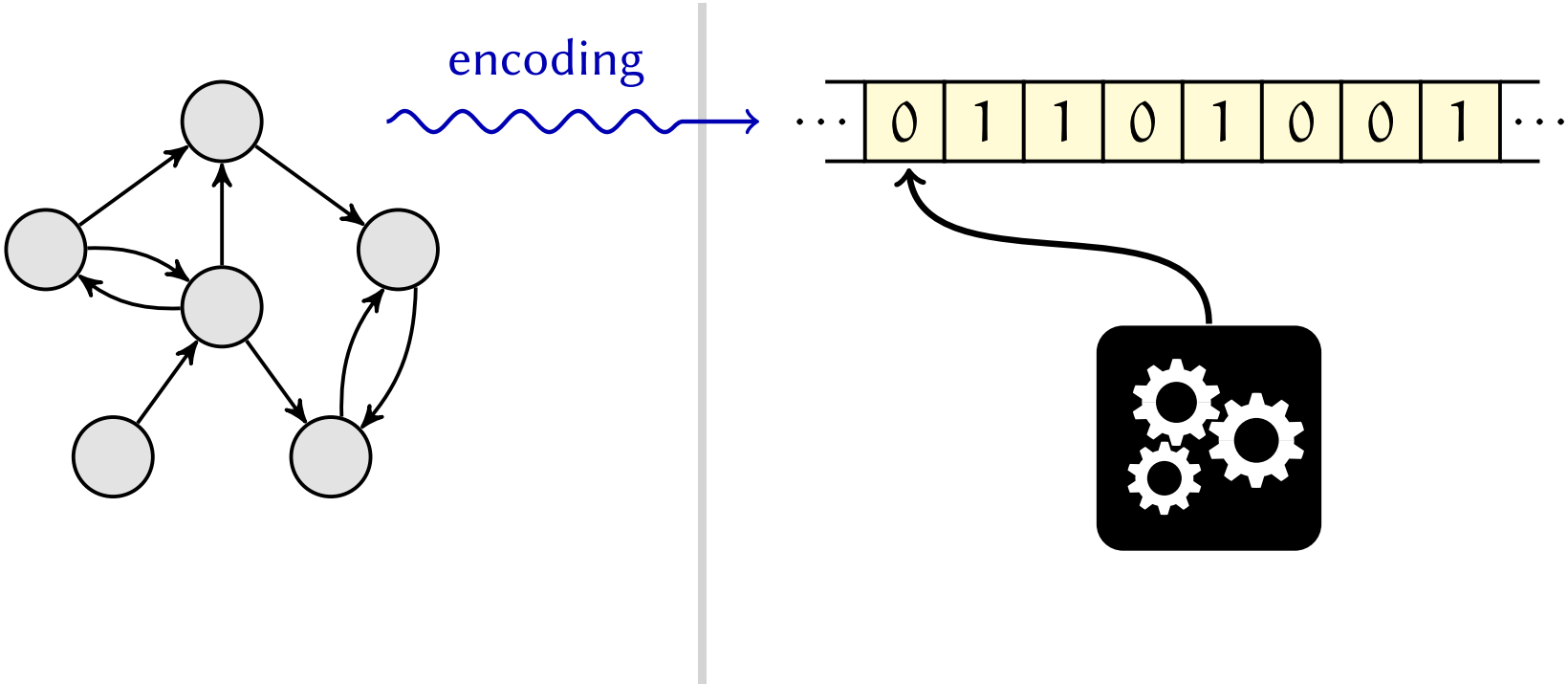


Example: Hamiltonian path

$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

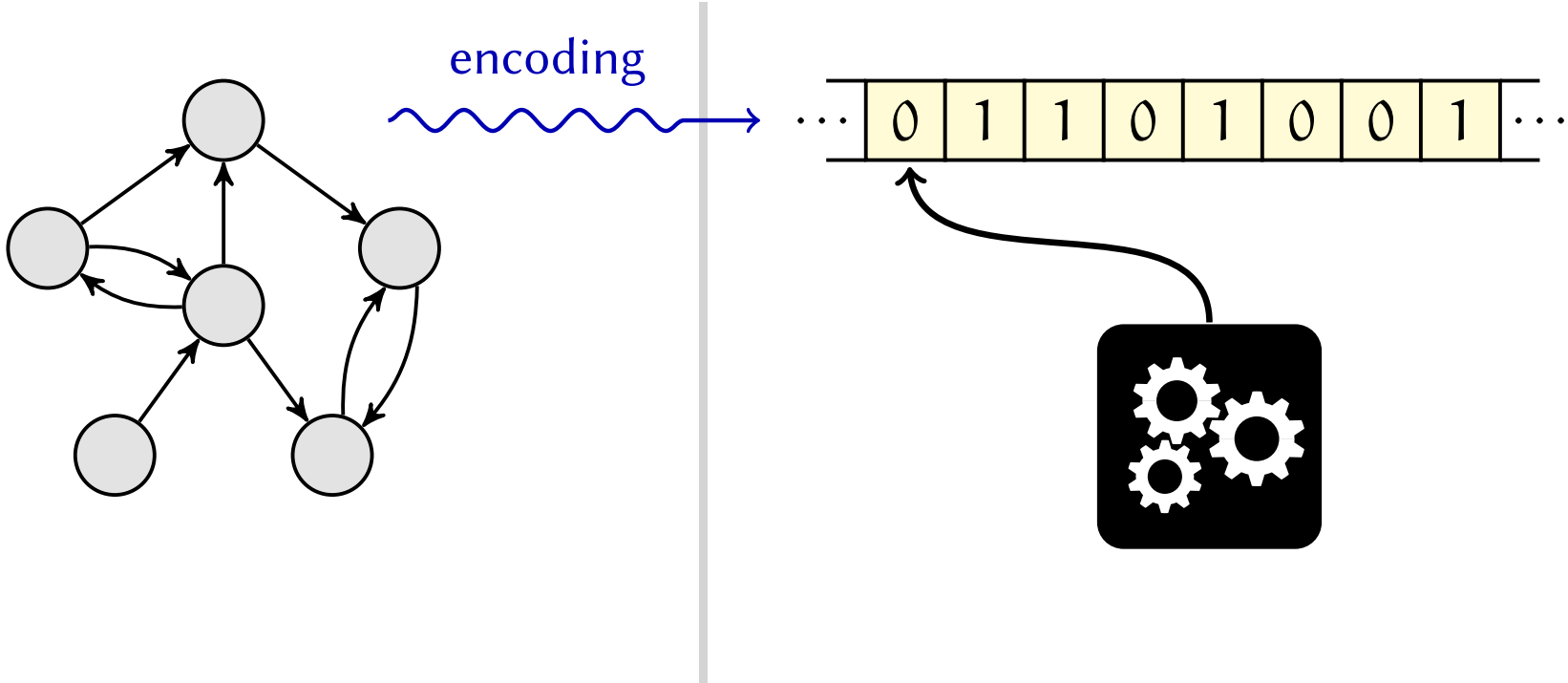
- Nondeterministic moves
- Polynomial running time

Descriptive complexity



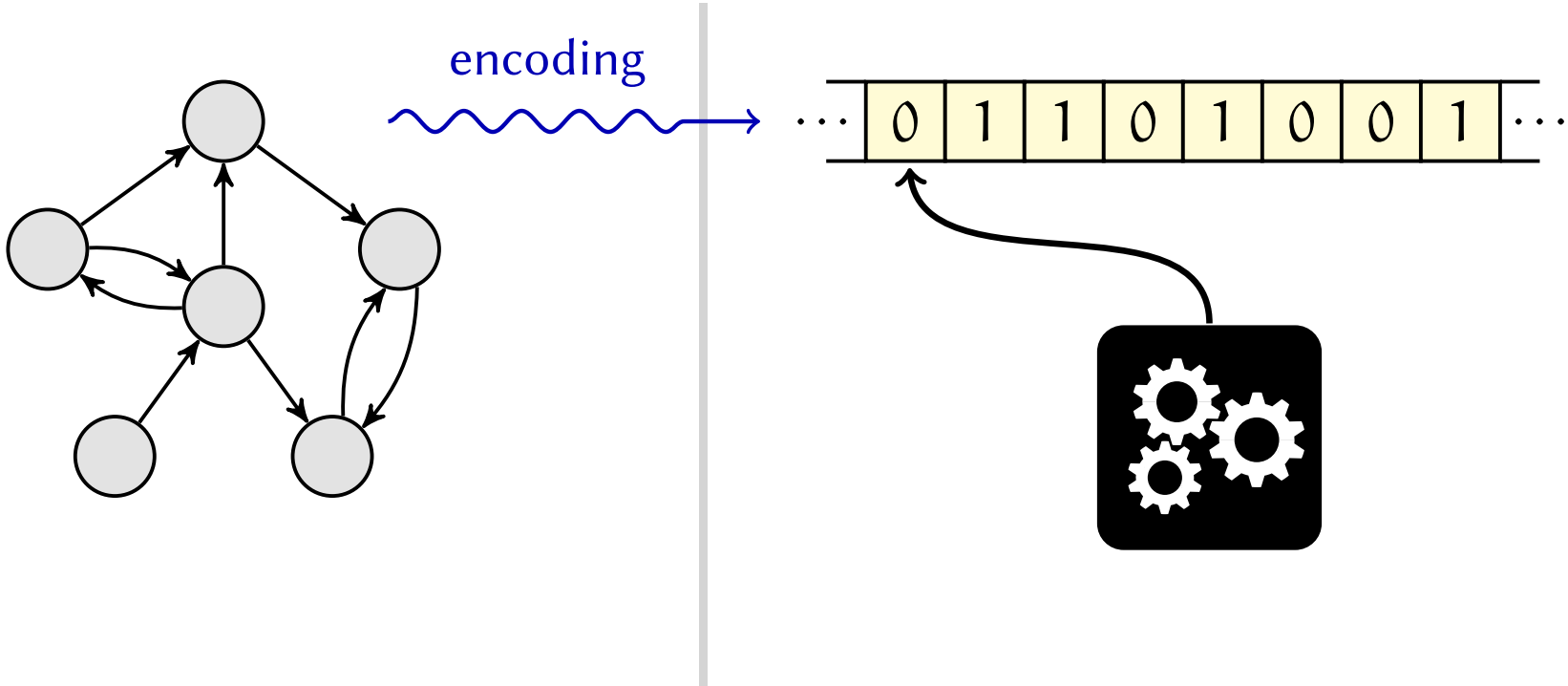
Descriptive complexity

SOME LOGICAL FORMALISM



Descriptive complexity

SOME LOGICAL FORMALISM

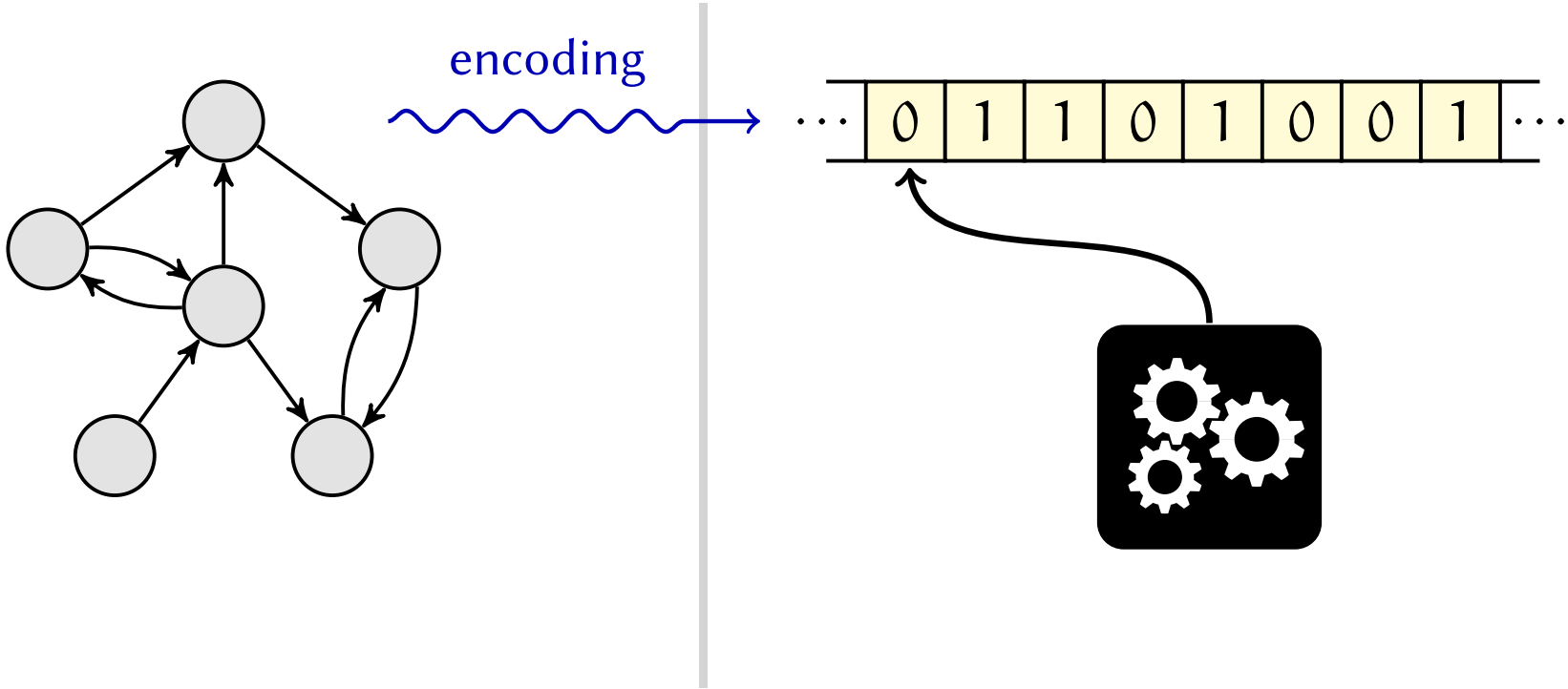


Descriptive complexity

SOME LOGICAL FORMALISM

EQUIVALENT

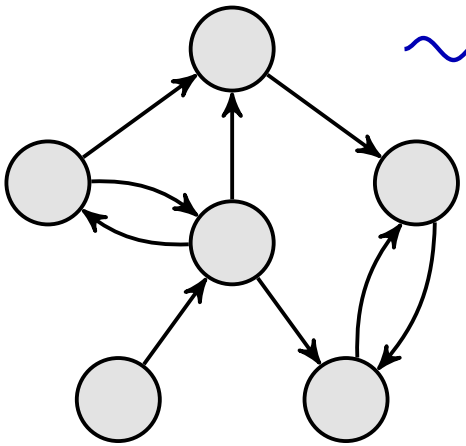
SOME ABSTRACT MACHINES



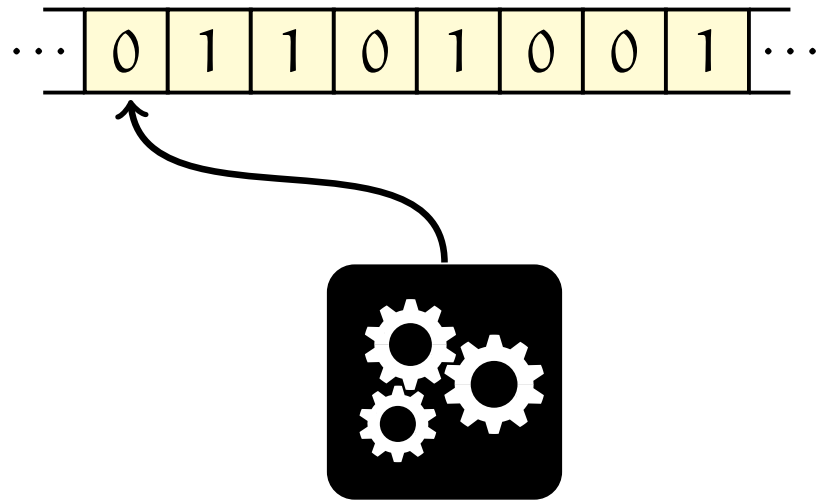
Descriptive complexity

SOME LOGICAL FORMALISM

SOME ABSTRACT MACHINES



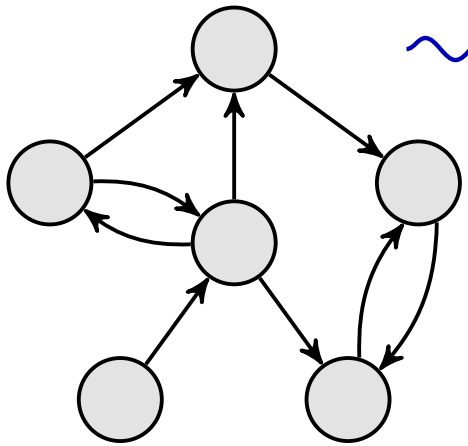
encoding



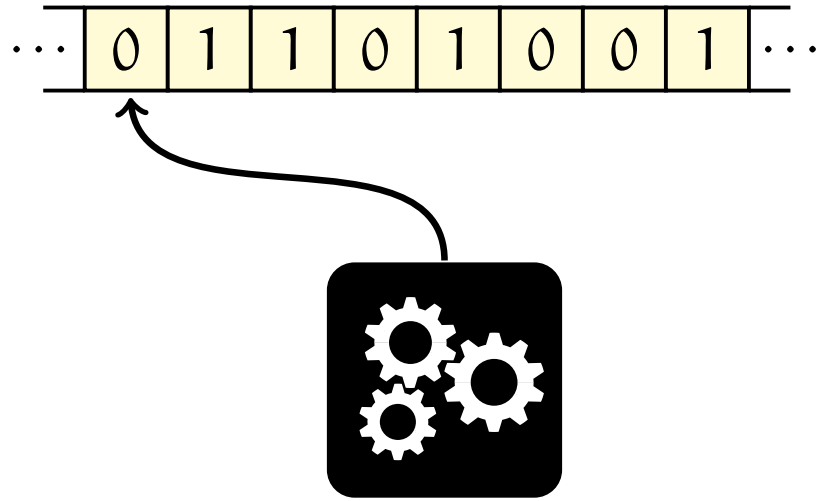
Formula class Φ

Descriptive complexity

SOME LOGICAL FORMALISM ⟺ EQUIVALENT ⟺ SOME ABSTRACT MACHINES



encoding



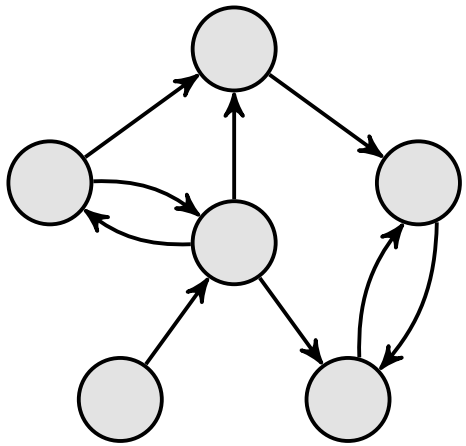
Formula class Φ

Algorithm class $\mathcal{A}$

Descriptive distributed complexity

Descriptive distributed complexity

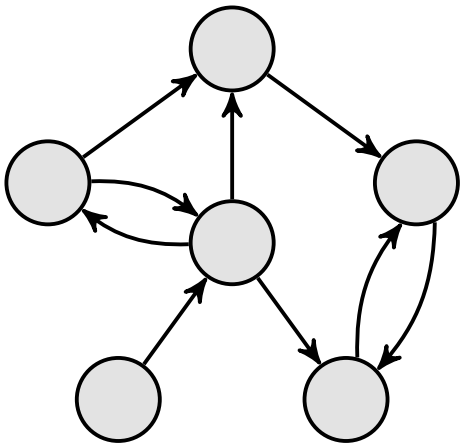
SOME LOGICAL FORMALISM



Formula class Φ

Descriptive distributed complexity

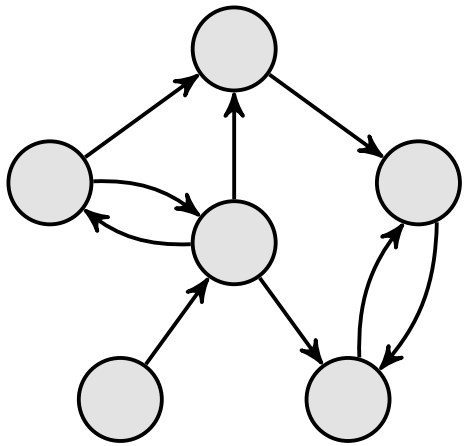
SOME LOGICAL FORMALISM



Formula class Φ

Descriptive distributed complexity

SOME LOGICAL FORMALISM ⟺ EQUIVALENT ⟺ COMMUNICATING MACHINES



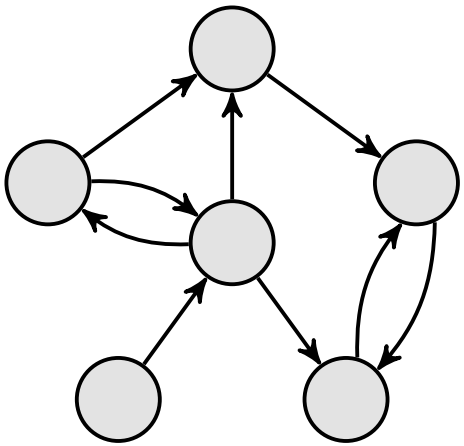
Formula class Φ

Descriptive distributed complexity

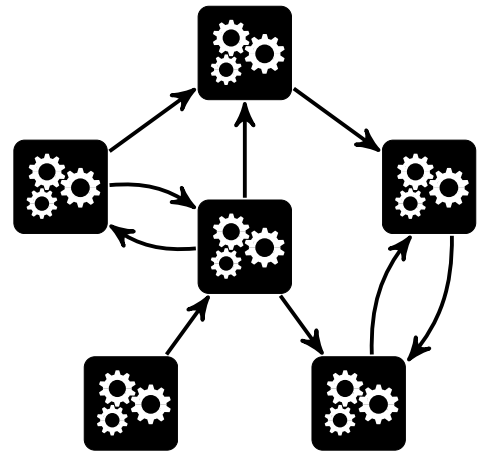
SOME LOGICAL FORMALISM

EQUIVALENT

COMMUNICATING MACHINES



Formula class Φ

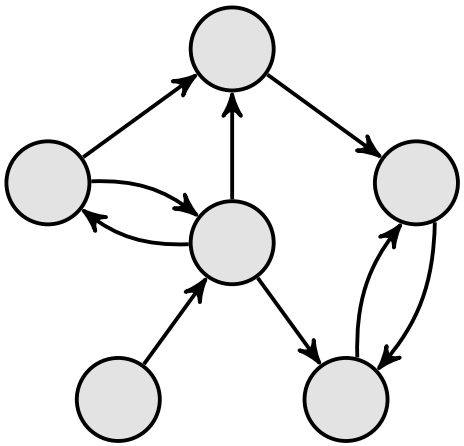


Descriptive distributed complexity

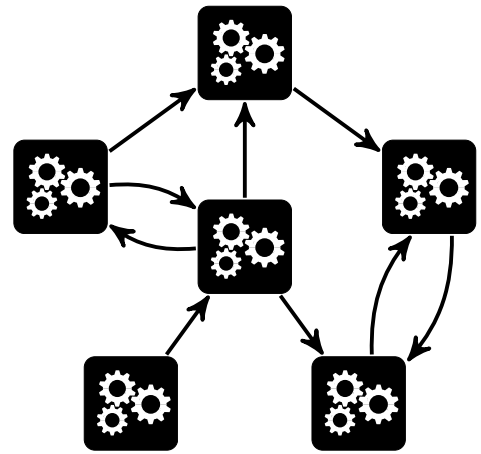
SOME LOGICAL FORMALISM

EQUIVALENT

COMMUNICATING MACHINES



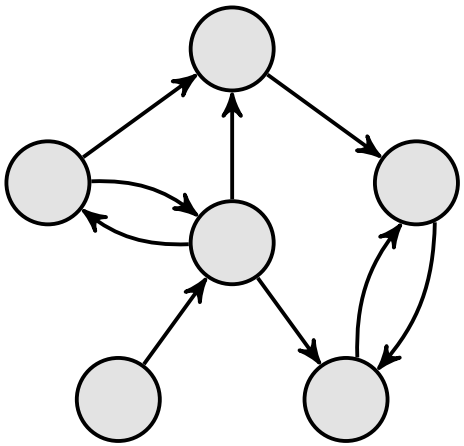
Formula class Φ



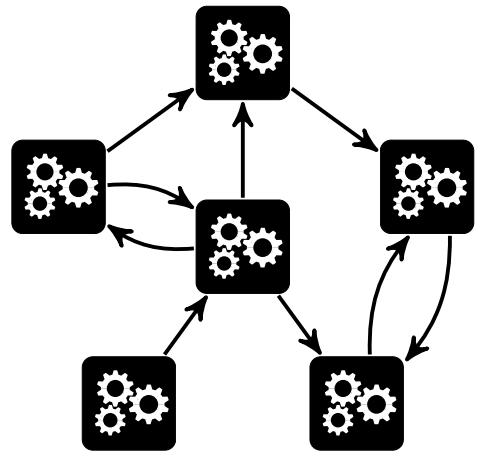
Distributed algorithm class $\mathcal{A}$

Descriptive distributed complexity

SOME LOGICAL FORMALISM ⟺ EQUIVALENT ⟺ COMMUNICATING MACHINES



Formula class Φ

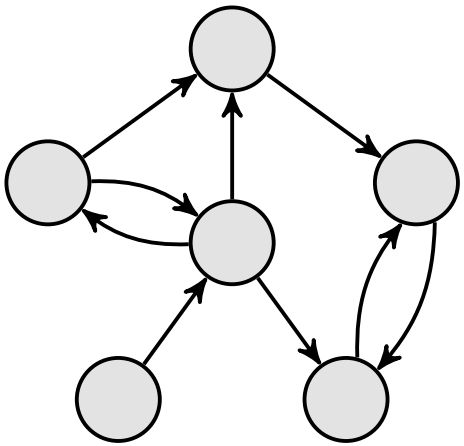


Distributed algorithm class $\mathcal{A}$

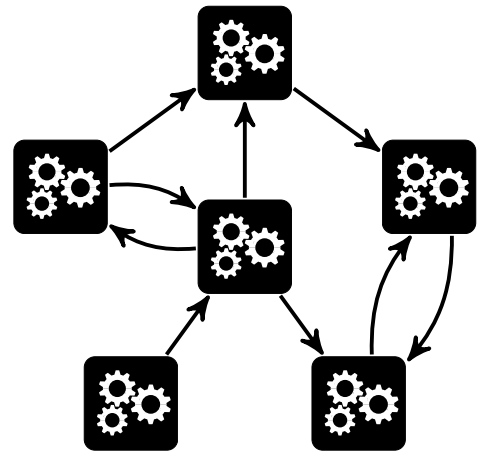
Unlike the sequential case:

Descriptive distributed complexity

SOME LOGICAL FORMALISM ⟺ EQUIVALENT ⟺ COMMUNICATING MACHINES



Formula class Φ



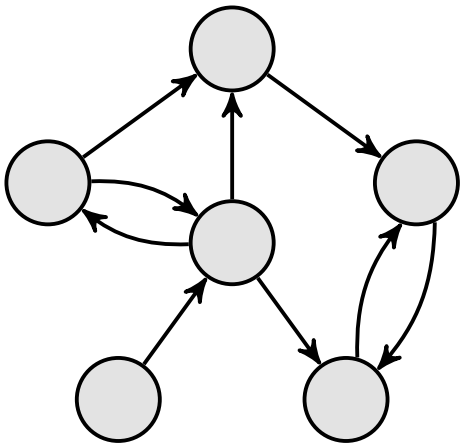
Distributed algorithm class $\mathcal{A}$

Unlike the sequential case:

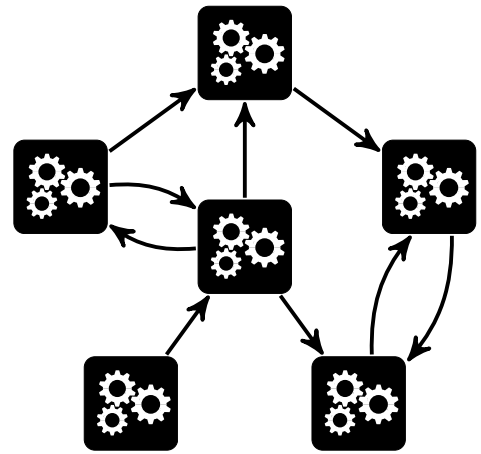
- The graph is not encoded.

Descriptive distributed complexity

SOME LOGICAL FORMALISM ⟺ EQUIVALENT ⟺ COMMUNICATING MACHINES



Formula class Φ

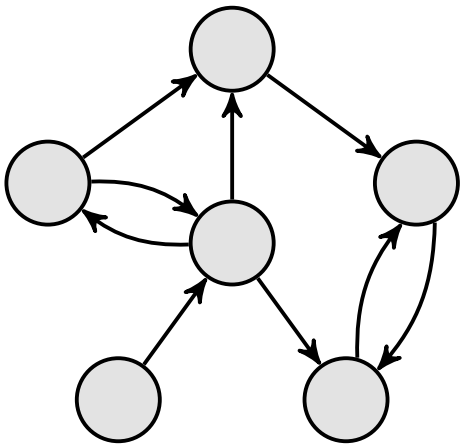


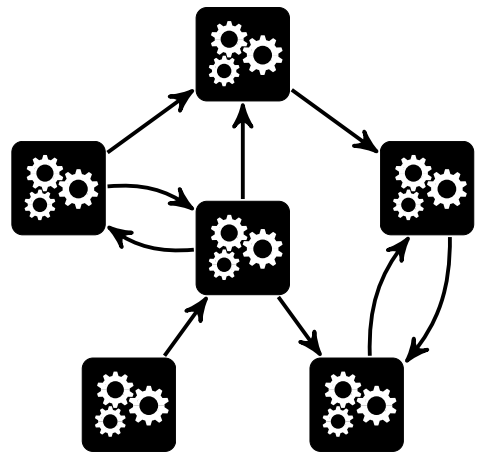
Distributed algorithm class $\mathcal{A}$

Unlike the sequential case:

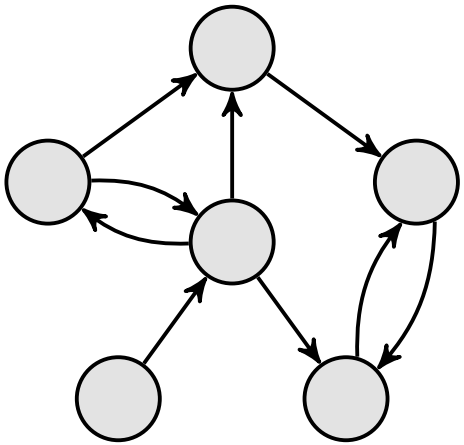
- The graph is not encoded.
- It does not have to be finite.

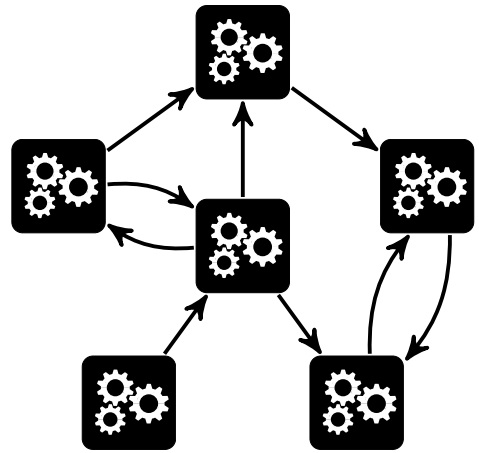
The “Helsinki-Tampere theorem” (2012)





The “Helsinki-Tampere theorem” (2012)

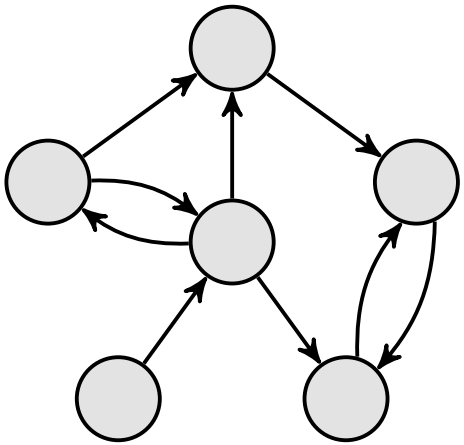


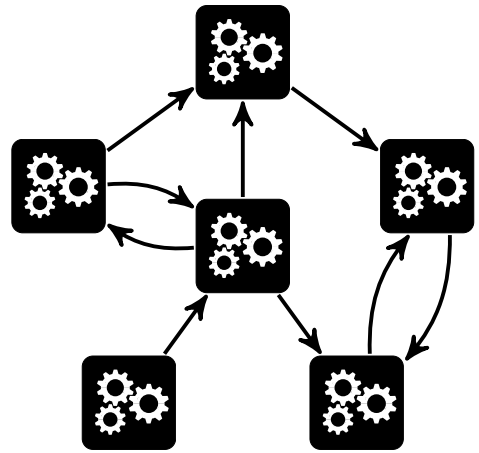


Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC

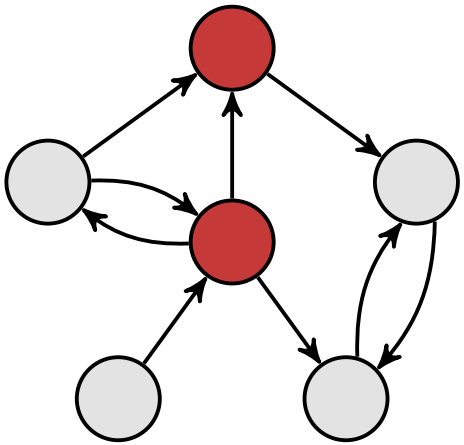


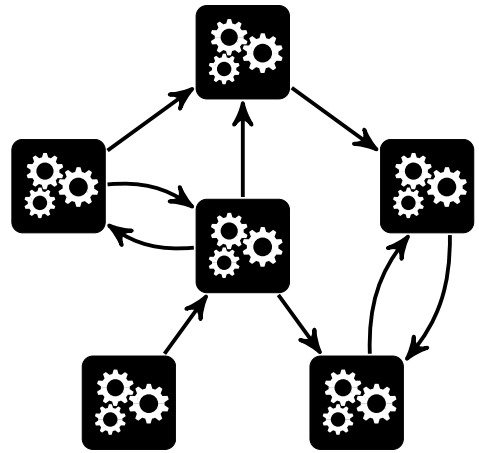


Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC

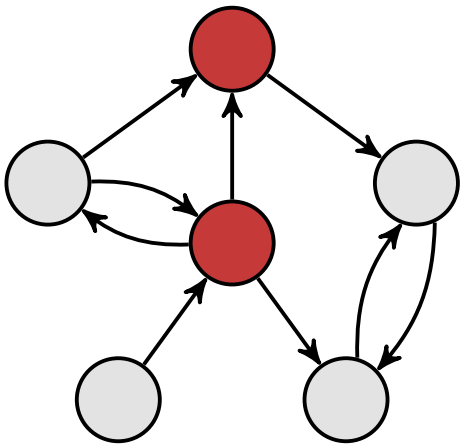


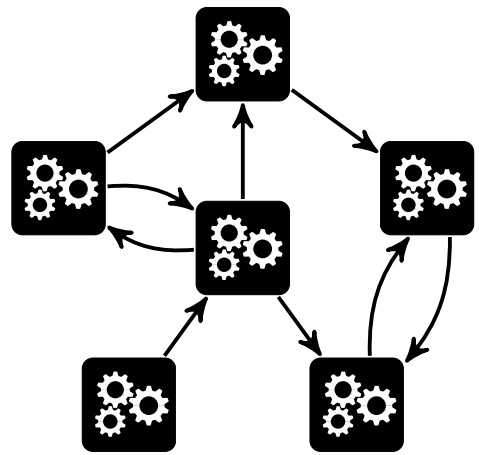


Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC



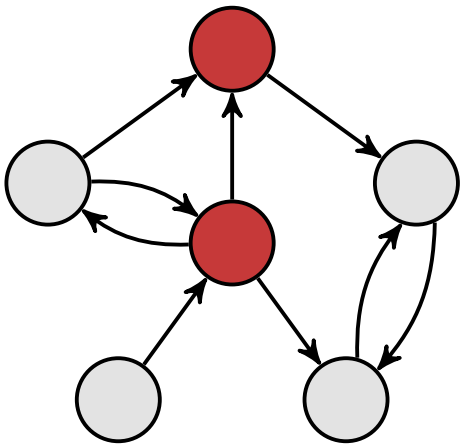


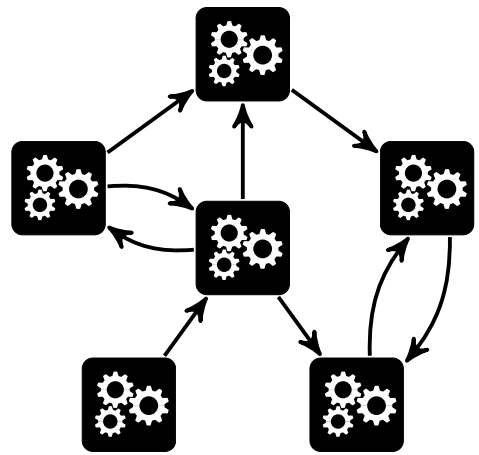
Example: $\overline{\Diamond}(\overline{\Box}\,\mathsf{white} \vee \overline{\Box}\,\mathsf{red})$

Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC





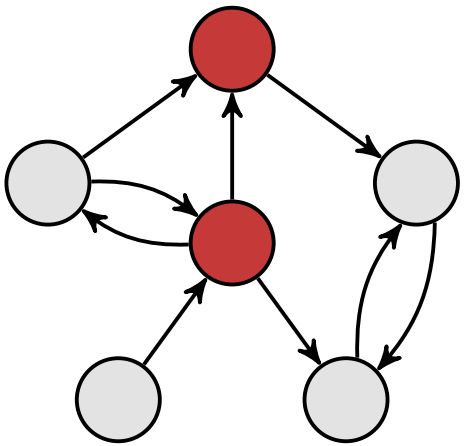
Example: $\overline{\Diamond}(\overline{\Box}\,\text{white} \vee \overline{\Box}\,\text{red})$

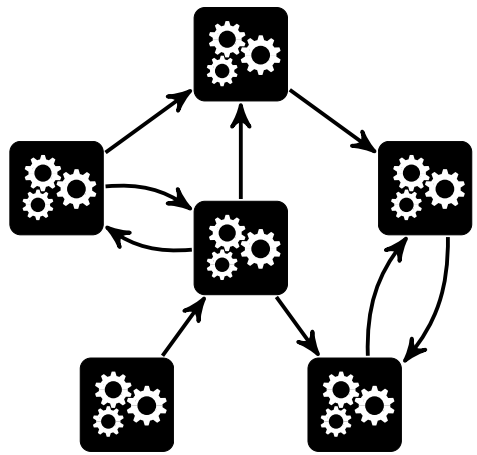
“I have an in-neighbor whose in-neighbors are all white or all red.”

Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC ⟺ EQUIVALENT ⟺ LOCAL DISTRIB. AUTOMATA





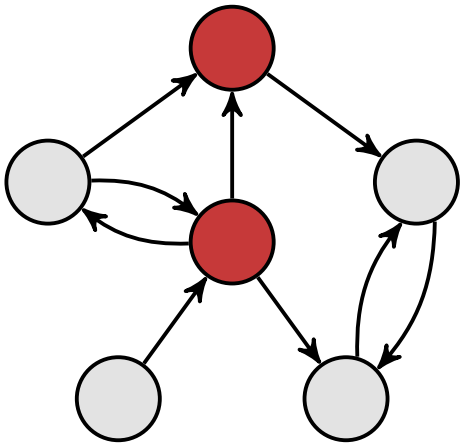
Example: $\overline{\Diamond}(\overline{\Box}\,\text{white} \vee \overline{\Box}\,\text{red})$

“I have an in-neighbor whose in-neighbors are all white or all red.”

Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

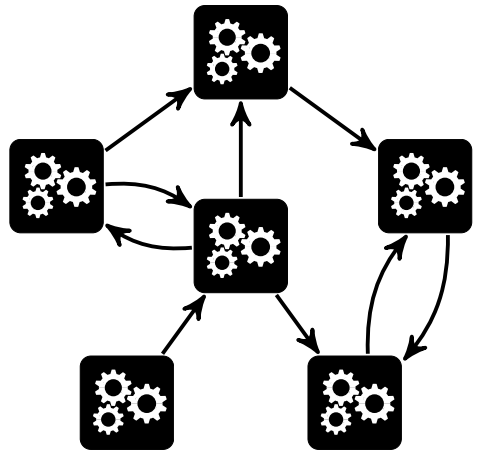
The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC ⟺ EQUIVALENT ⟺ LOCAL DISTRIB. AUTOMATA



Example: $\overline{\Diamond}(\overline{\Box}\,\text{white} \vee \overline{\Box}\,\text{red})$

“I have an in-neighbor whose in-neighbors are all white or all red.”



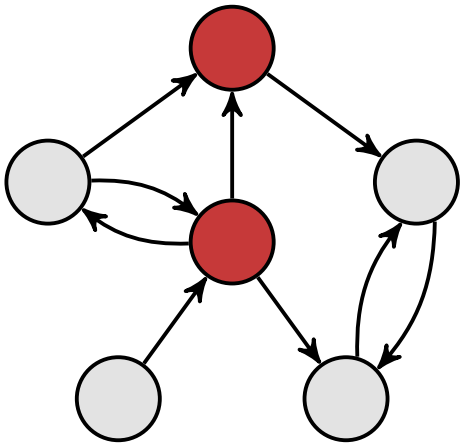
 : Finite-state machine

$\delta\colon Q \times 2^Q \to Q$

Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

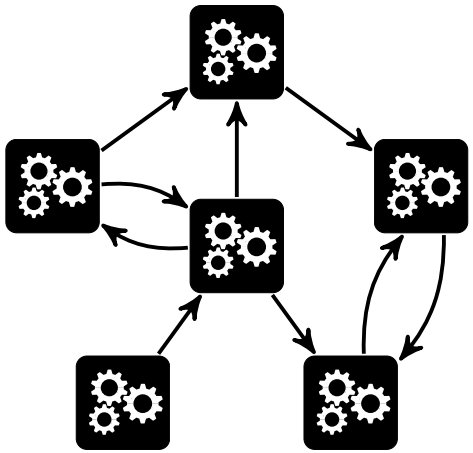
The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC ⟺ EQUIVALENT ⟺ LOCAL DISTRIB. AUTOMATA



Example: $\overline{\Diamond}(\overline{\Box}\,\text{white} \vee \overline{\Box}\,\text{red})$

“I have an in-neighbor whose in-neighbors are all white or all red.”



 : Finite-state machine

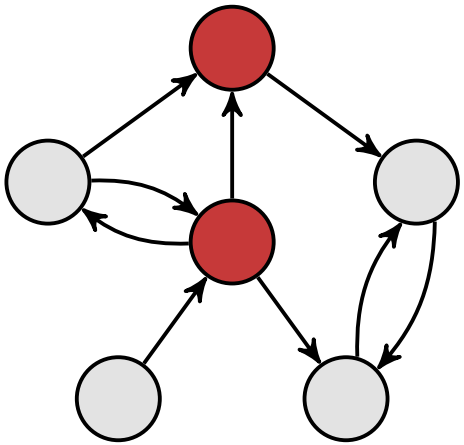
$\delta\colon Q \times 2^Q \to Q$

- Synchronous execution

Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

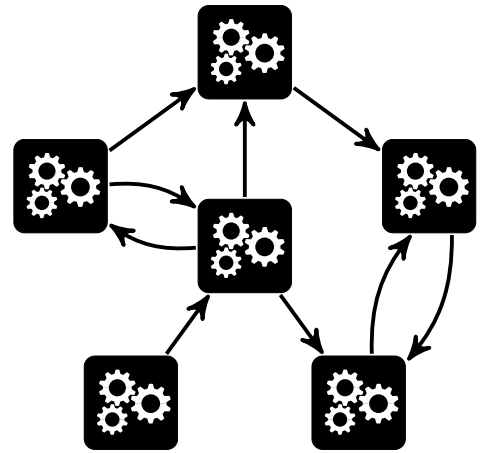
The “Helsinki-Tampere theorem” (2012)

BACKWARD MODAL LOGIC ⟺ EQUIVALENT ⟺ LOCAL DISTRIB. AUTOMATA



Example: $\overline{\Diamond}(\overline{\Box}\,\text{white} \vee \overline{\Box}\,\text{red})$

“I have an in-neighbor whose in-neighbors are all white or all red.”



 : Finite-state machine

$\delta\colon Q \times 2^Q \to Q$

- Synchronous execution
- Constant running time

Hella · Järvisalo · Kuusisto · Laurinharju · Lempiäinen · Luosto · Suomela · Virtema

Contributions

Contributions

MONADIC SECOND-ORDER LOGIC

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

\+ Alternation

Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

- \+ Alternation
- \+ Global acceptance

Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

- \+ Alternation
- \+ Global acceptance

THE BACKWARD μ-FRAGMENT

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

- \+ Alternation
- \+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

Contributions

MONADIC SECOND-ORDER LOGIC

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

+ Alternation
+ Global acceptance

THE BACKWARD μ-FRAGMENT

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

Contributions

MONADIC SECOND-ORDER LOGIC

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

+ Alternation
+ Global acceptance

THE BACKWARD μ-FRAGMENT

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

$$\delta \colon Q \times 2^Q \rightarrow Q$$

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

+ Alternation
+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \rightarrow Q$$

+ Unbounded running time

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

- \+ Alternation
- \+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \rightarrow Q$$

- \+ Unbounded running time
- − Asynchronous execution

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

+ Alternation
+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \rightarrow Q$$

+ Unbounded running time
− Asynchronous execution

Other contributions:

Contributions

MONADIC SECOND-ORDER LOGIC

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

$$\delta \colon Q \times 2^Q \rightarrow Q$$

\+ Unbounded running time

− Asynchronous execution

Other contributions:

- Emptiness problems for deterministic nonlocal automata.

Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \to 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

$$\delta \colon Q \times 2^Q \to Q$$

\+ Unbounded running time

− Asynchronous execution

Other contributions:

- Emptiness problems for deterministic nonlocal automata.
- Connections to classical automata on words and trees.

Contributions

MONADIC SECOND-ORDER LOGIC

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\forall Z\left(\exists x, y\big(Z(x) \wedge \neg Z(y)\big) \to \cdots\right)$$

$$\delta\colon Q \times 2^Q \to 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\mu\begin{pmatrix} X \\ Y \end{pmatrix}.\begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

$$\delta\colon Q \times 2^Q \to Q$$

\+ Unbounded running time

− Asynchronous execution

Other contributions:

- Emptiness problems for deterministic nonlocal automata.
- Connections to classical automata on words and trees.
- Set quantifier alternation hierarchies in modal logic.

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

- \+ Alternation
- \+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \rightarrow Q$$

- \+ Unbounded running time
- − Asynchronous execution

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \rightarrow Q$$

\+ Unbounded running time

− Asynchronous execution

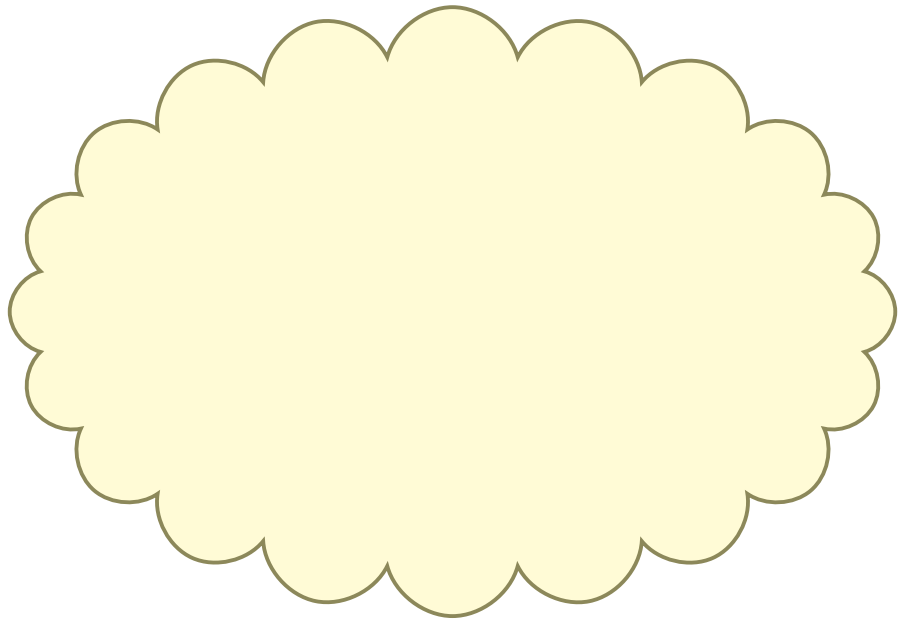
Monadic second-order logic (MSOL)

Monadic second-order logic (MSOL)

Example: weakly connected digraph

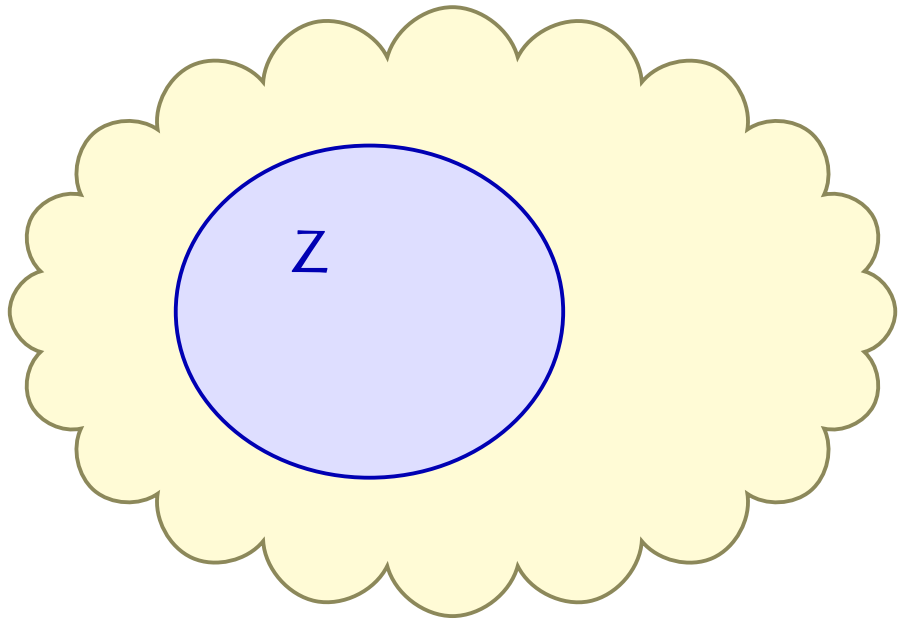
Monadic second-order logic (MSOL)

Example: weakly connected digraph



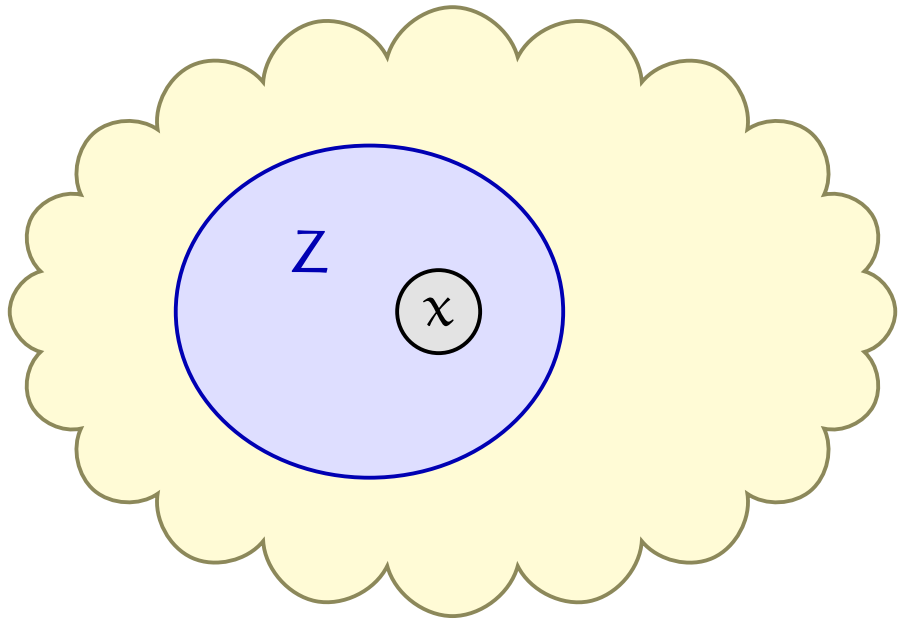
Monadic second-order logic (MSOL)

Example: weakly connected digraph



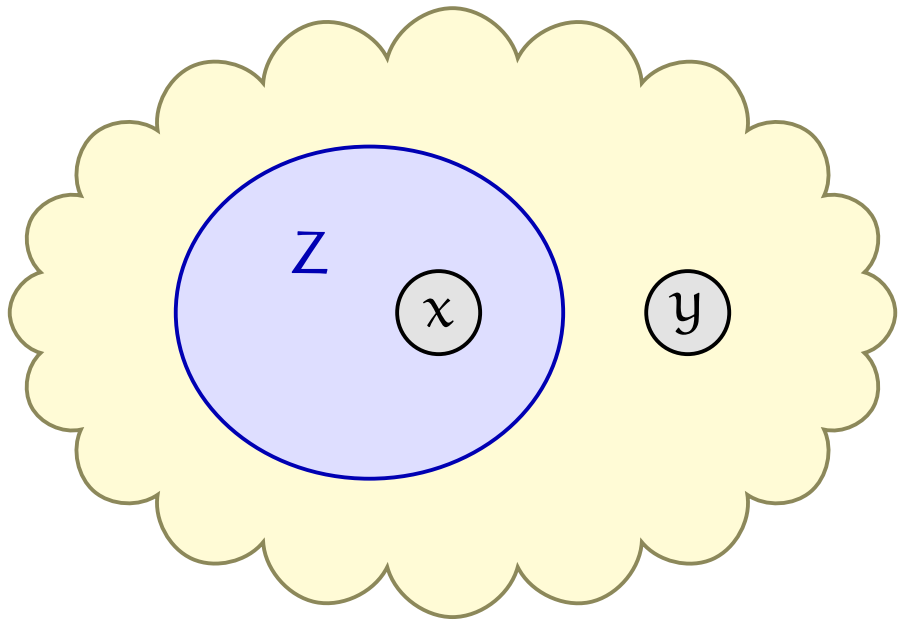
Monadic second-order logic (MSOL)

Example: weakly connected digraph



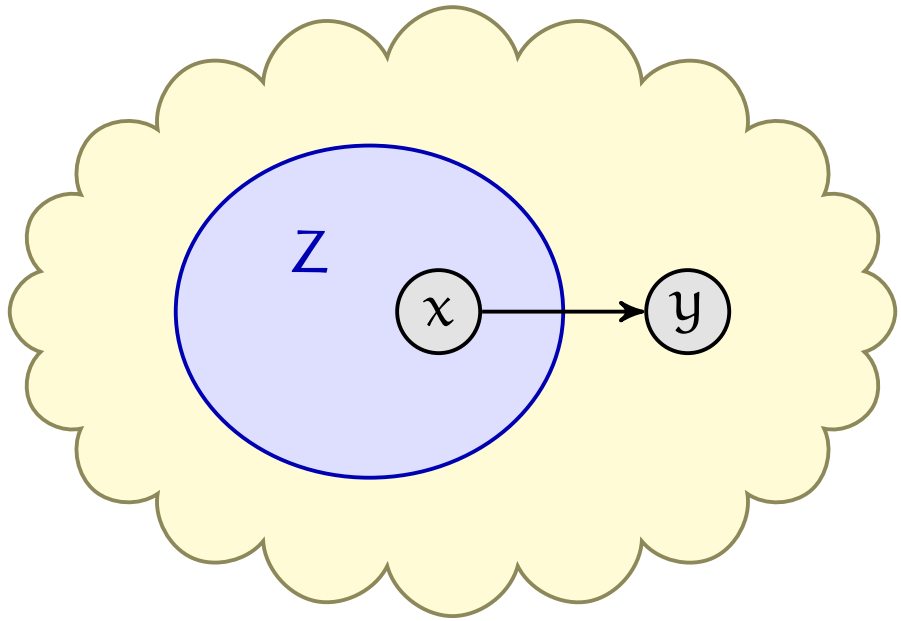
Monadic second-order logic (MSOL)

Example: weakly connected digraph



Monadic second-order logic (MSOL)

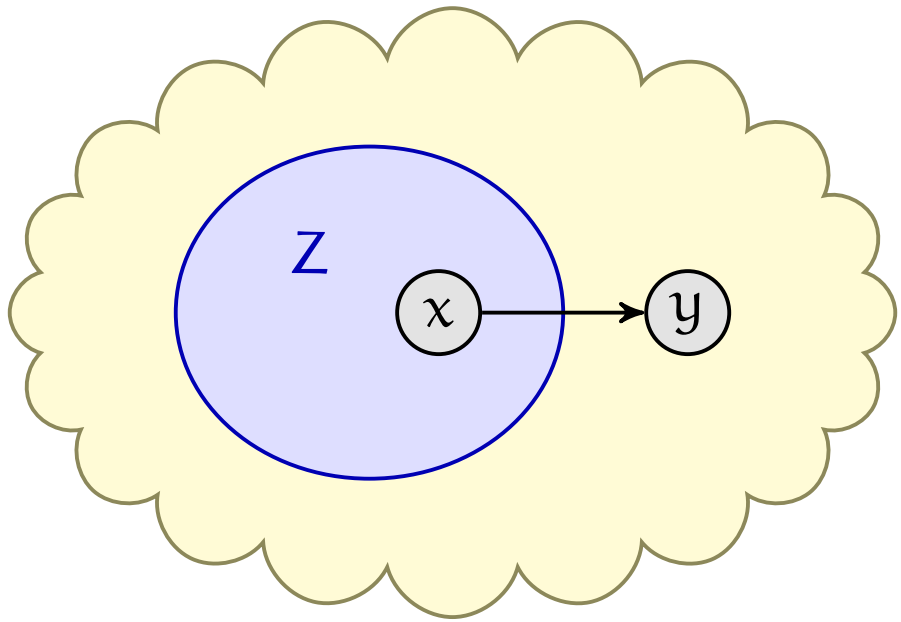
Example: weakly connected digraph



Monadic second-order logic (MSOL)

Example: weakly connected digraph

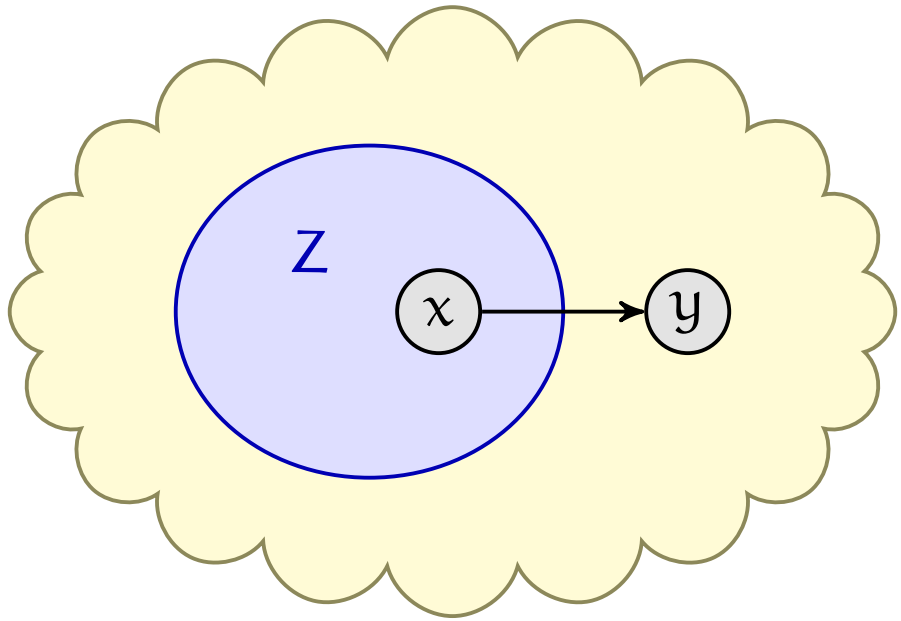
$\forall Z ($ $)$



Monadic second-order logic (MSOL)

Example: weakly connected digraph

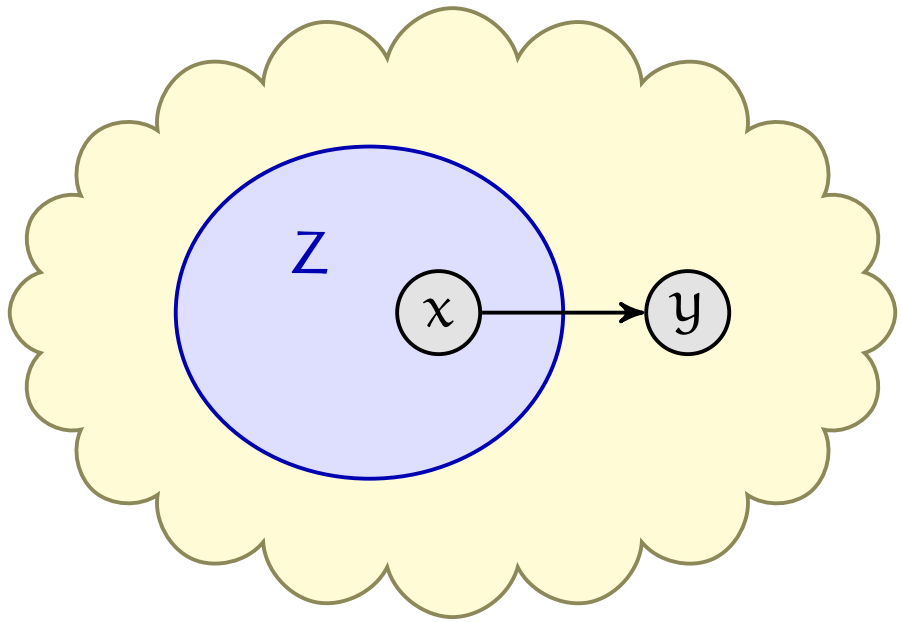
$$\forall Z \Big(\underbrace{\qquad\qquad\qquad\qquad}_{Z \text{ is a nontrivial subset.}} \Big)$$



Monadic second-order logic (MSOL)

Example: weakly connected digraph

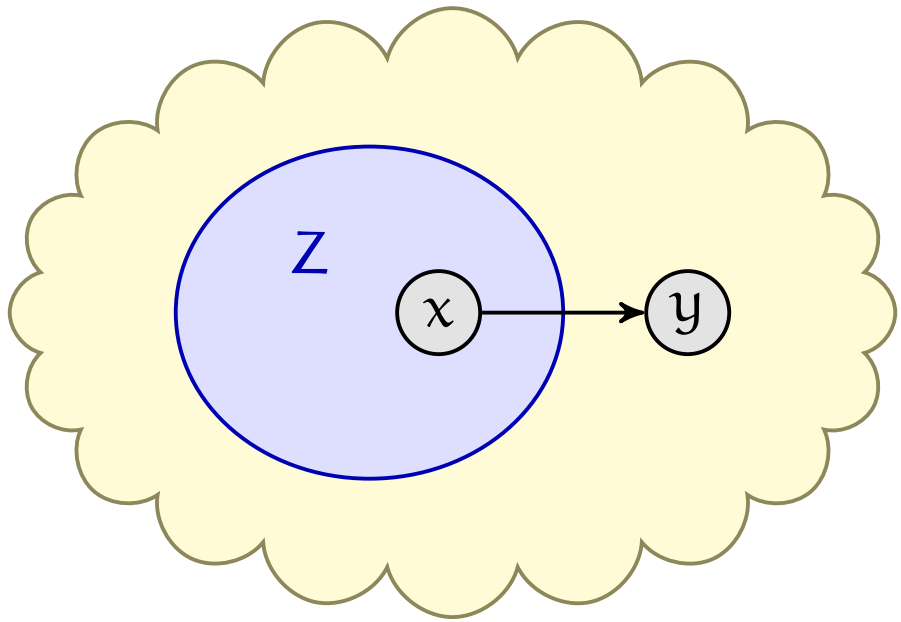
$$\forall Z \Big(\underbrace{\qquad\qquad}_{Z \text{ is a nontrivial subset.}} \rightarrow \underbrace{\qquad\qquad\qquad\qquad}_{Z \text{ is connected to its complement.}} \Big)$$



Monadic second-order logic (MSOL)

Example: weakly connected digraph

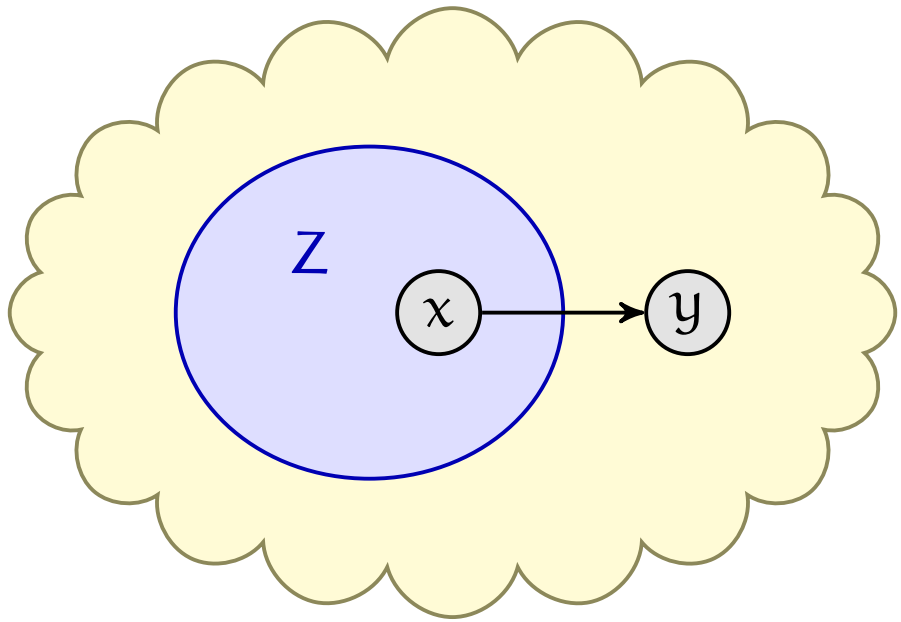
$$\forall Z \Big(\underbrace{\exists x, y \big(Z(x) \wedge \neg Z(y) \big)}_{\text{Z is a nontrivial subset.}} \rightarrow \underbrace{\qquad\qquad\qquad\qquad}_{\text{Z is connected to its complement.}} \Big)$$



Monadic second-order logic (MSOL)

Example: weakly connected digraph

$$\forall Z \Big(\underbrace{\exists x, y \big(Z(x) \wedge \neg Z(y) \big)}_{\text{Z is a nontrivial subset.}} \rightarrow \underbrace{\exists x, y \big((Z(x) \leftrightarrow \neg Z(y)) \wedge E(x,y) \big)}_{\text{Z is connected to its complement.}} \Big)$$



Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \to \cdots \right)$$

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \to 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \to Q$$

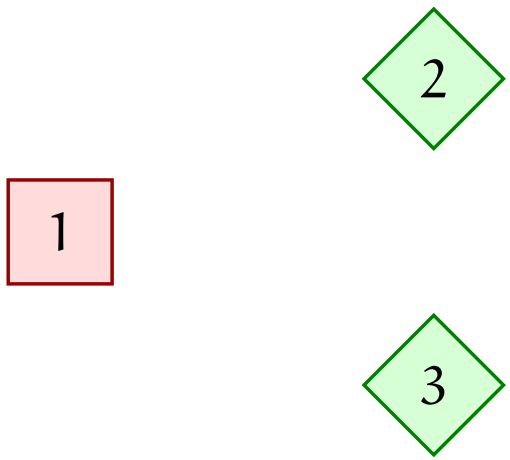
\+ Unbounded running time

− Asynchronous execution

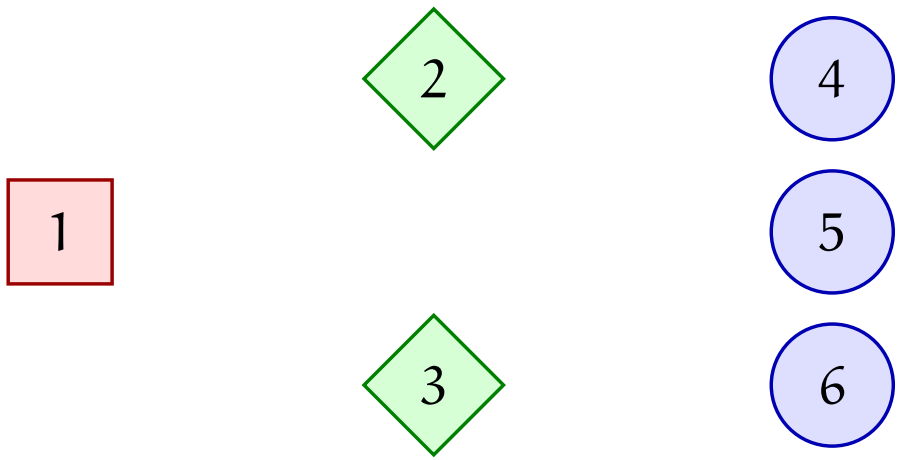
Alternating local automata

Alternating local automata

Alternating local automata

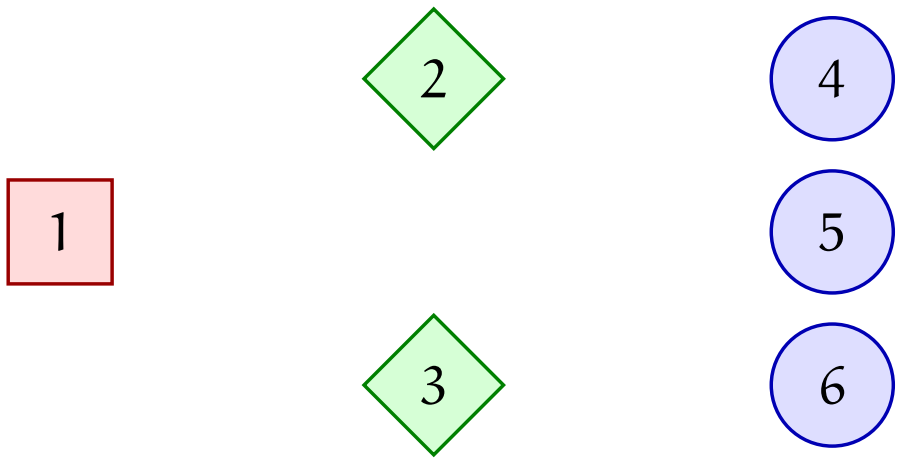


Alternating local automata

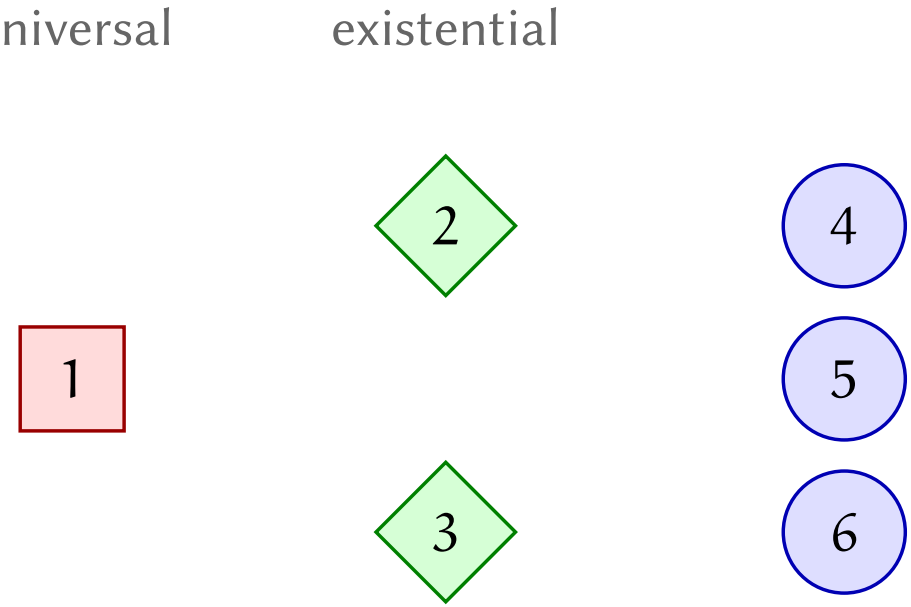


Alternating local automata

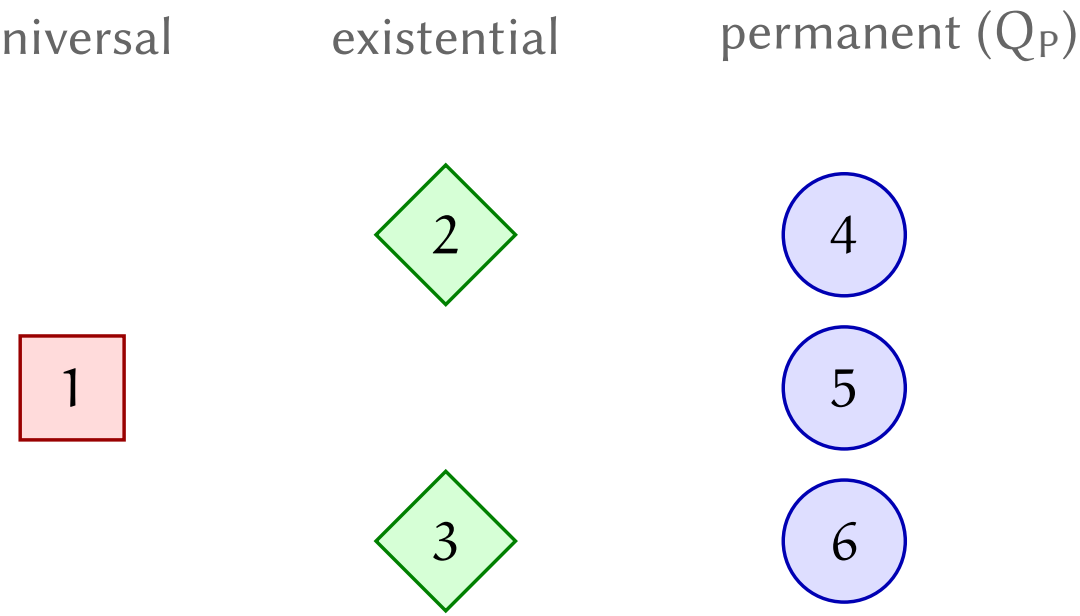
universal



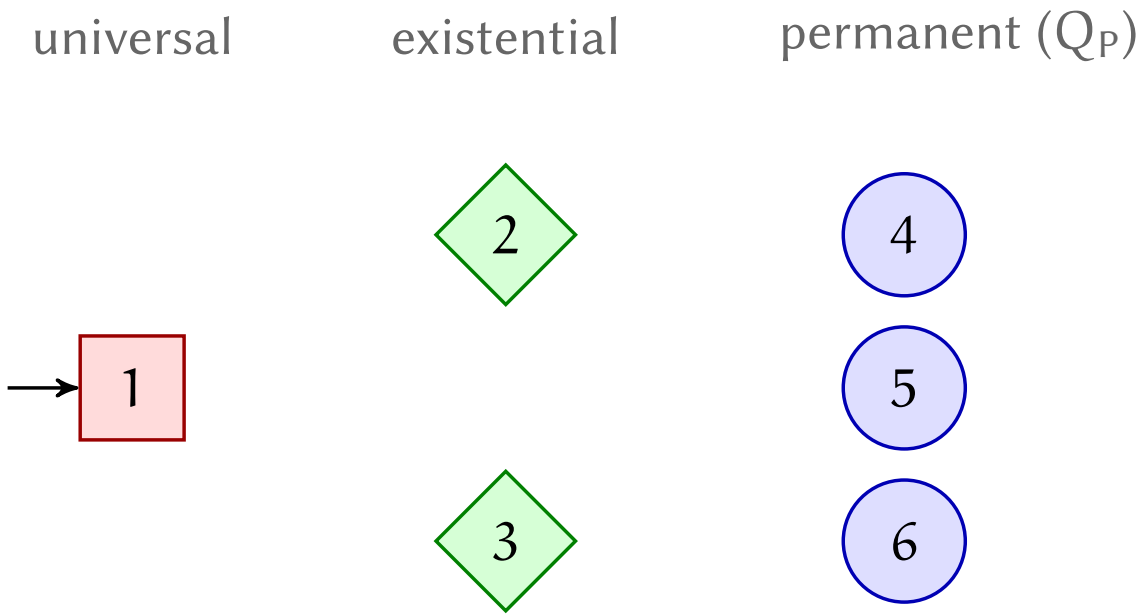
Alternating local automata



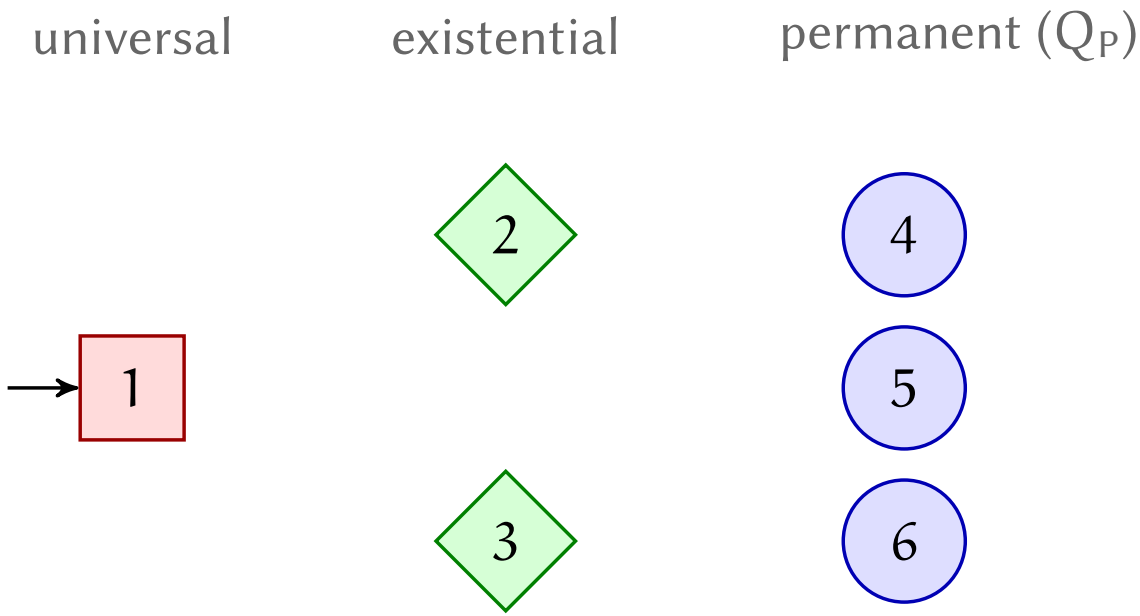
Alternating local automata



Alternating local automata



Alternating local automata

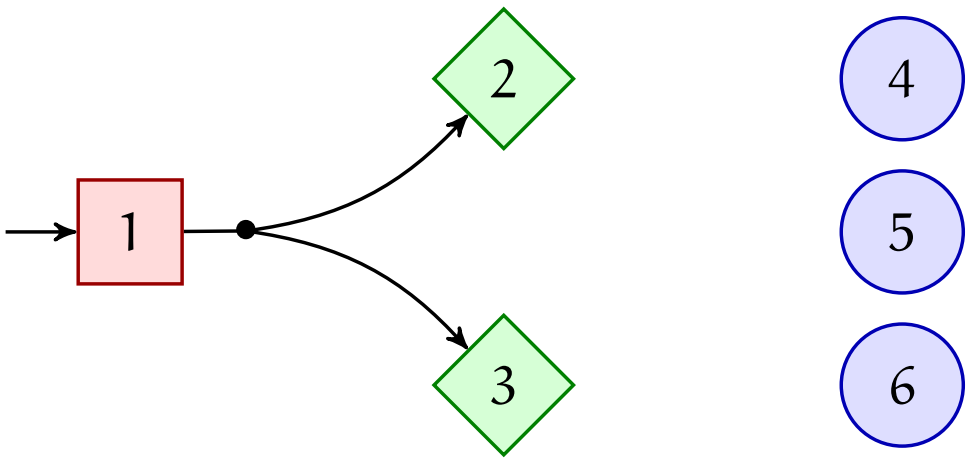


$$\delta\colon Q \times 2^Q \to 2^Q$$

(transition)

Alternating local automata

universal existential permanent (Q_P)

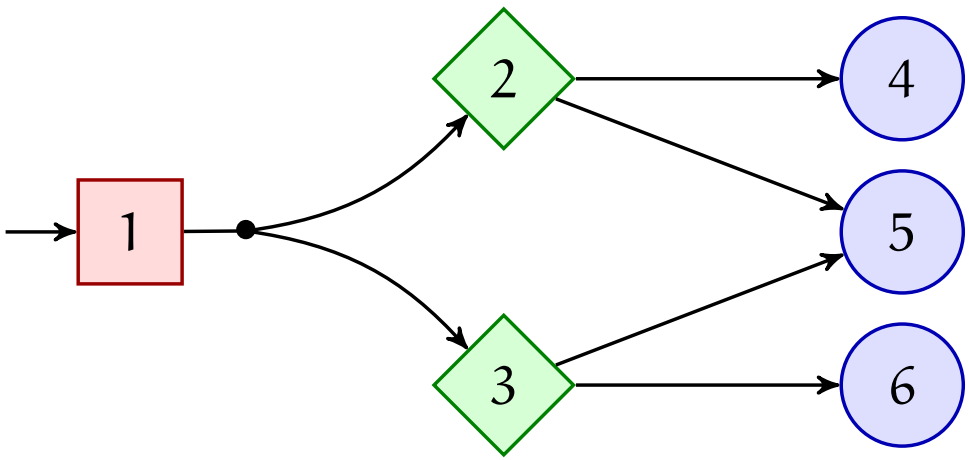


$$\delta: Q \times 2^Q \to 2^Q$$

(transition)

Alternating local automata

universal existential permanent (Q_P)

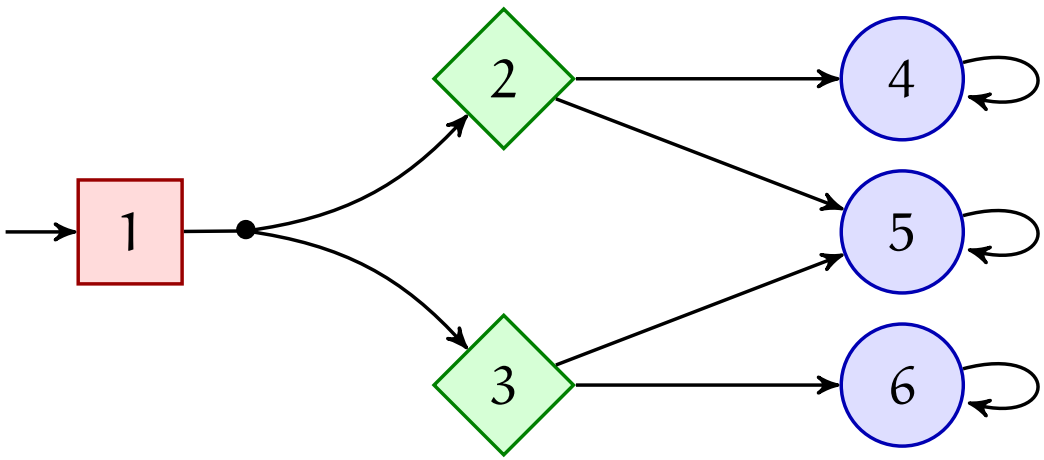


$$\delta: Q \times 2^Q \to 2^Q$$

(transition)

Alternating local automata

universal existential permanent (Q_P)

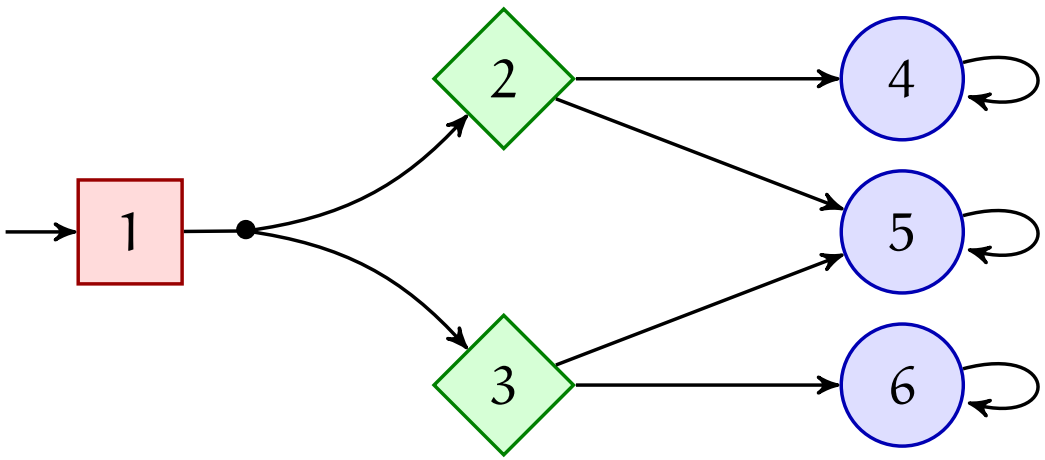


$\delta: Q \times 2^Q \to 2^Q$

(transition)

Alternating local automata

universal existential permanent (Q_P)



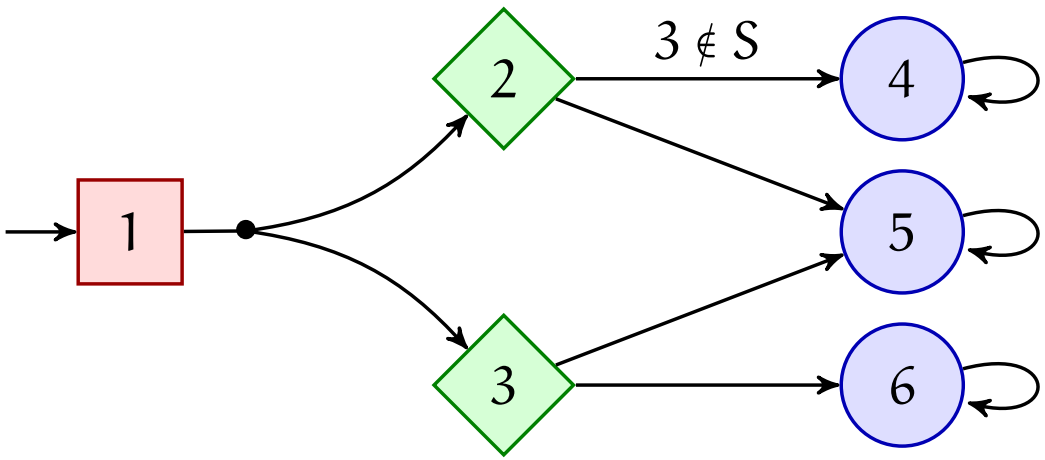
S: *set of received states*

$$\delta: Q \times 2^Q \to 2^Q$$

(transition)

Alternating local automata

universal existential permanent (Q_P)



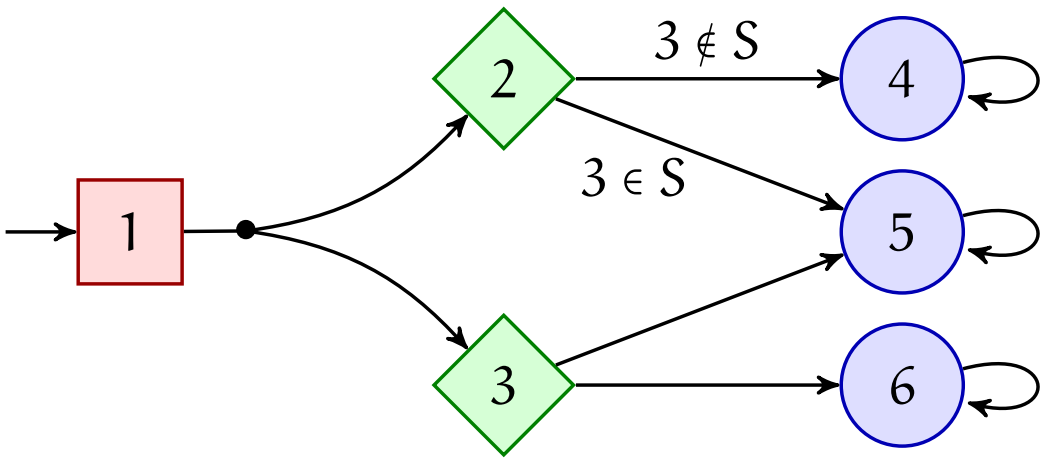
S: *set of received states*

$$\delta: Q \times 2^Q \to 2^Q$$

(transition)

Alternating local automata

universal existential permanent (Q_P)



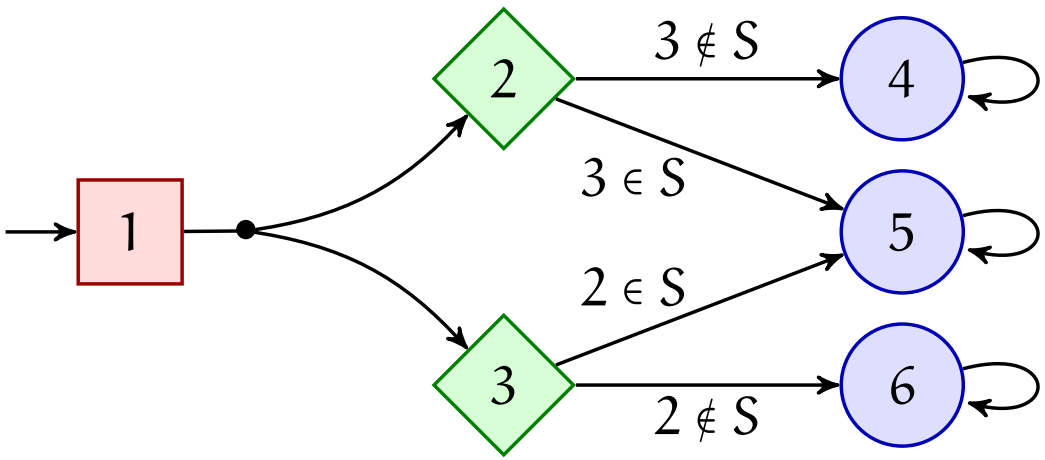
S: *set of received states*

$$\delta: Q \times 2^Q \to 2^Q$$

(transition)

Alternating local automata

universal existential permanent (Q_P)



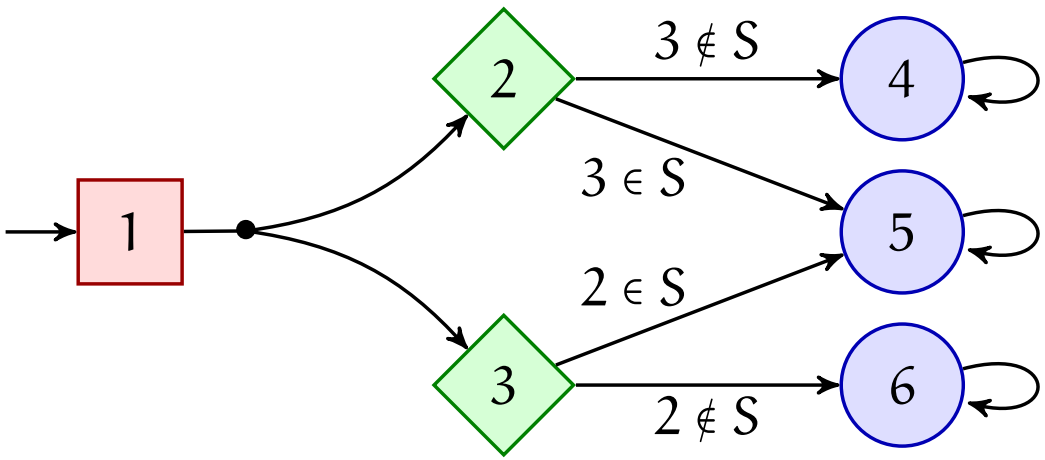
S: *set of received states*

$\delta: Q \times 2^Q \to 2^Q$

(transition)

Alternating local automata

universal existential permanent (Q_P)



S: *set of received states*

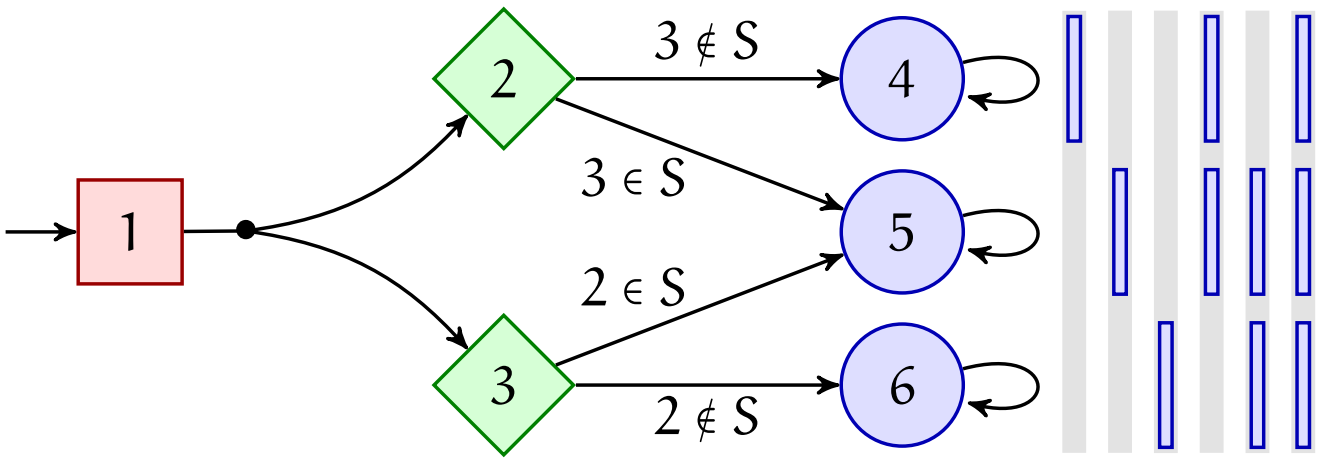
$\delta: Q \times 2^Q \to 2^Q$

(transition)

$\mathcal{F} \subseteq 2^{Q_P}$

(global acceptance)

Alternating local automata



S: *set of received states*

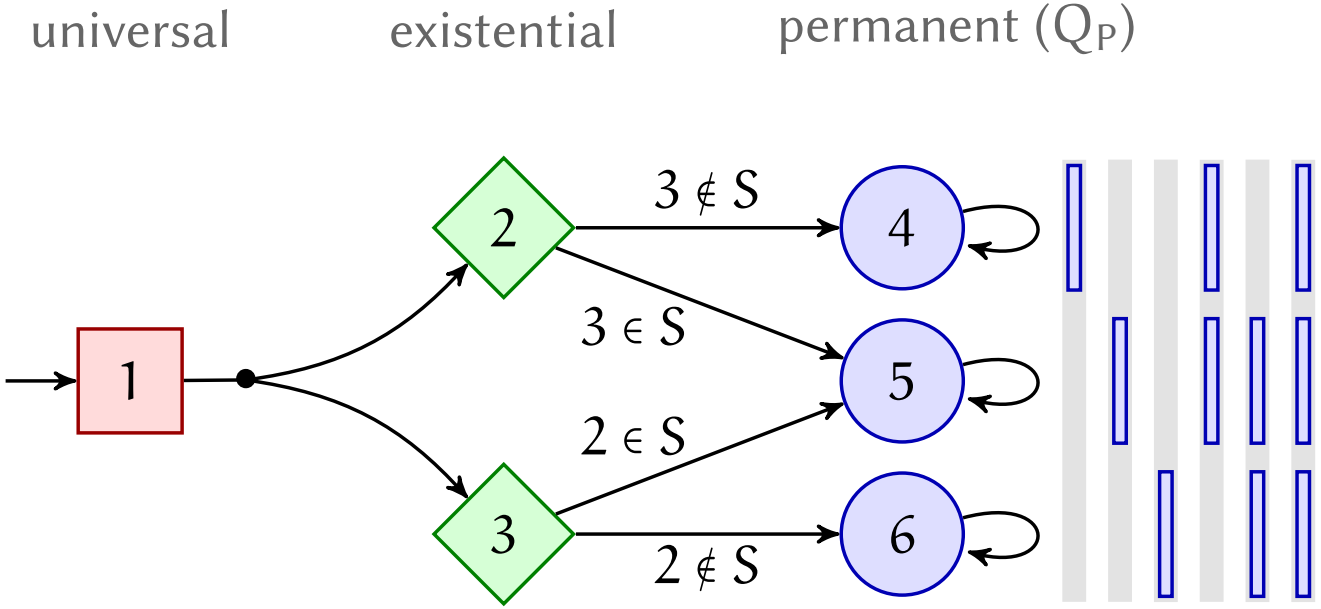
$\delta: Q \times 2^Q \to 2^Q$

(transition)

$\mathcal{F} \subseteq 2^{Q_P}$

(global acceptance)

Alternating local automata



S: *set of received states*

$\delta: Q \times 2^Q \to 2^Q$

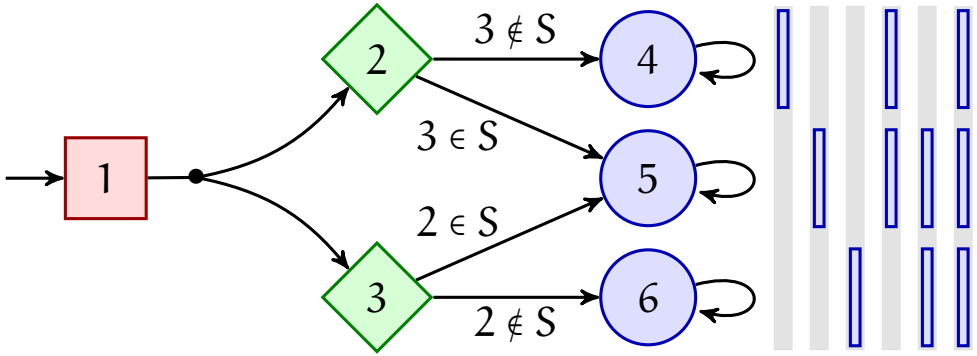
(transition)

$\mathcal{F} \subseteq 2^{Q_P}$

(global acceptance)

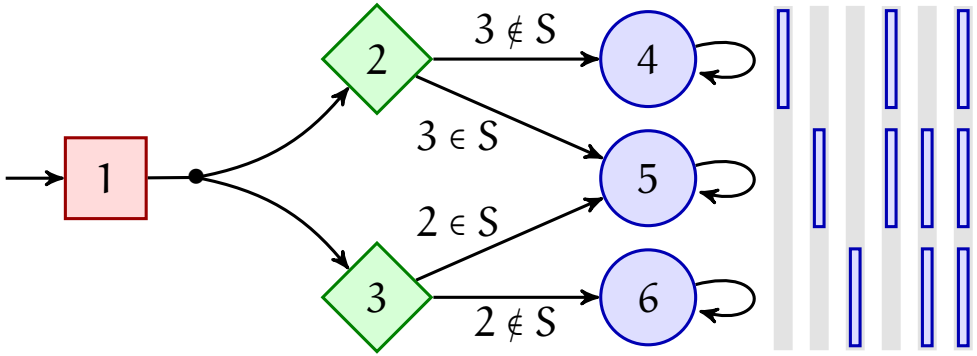
Same example: weakly connected digraph

Alternating run

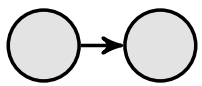


S: *set of received states*

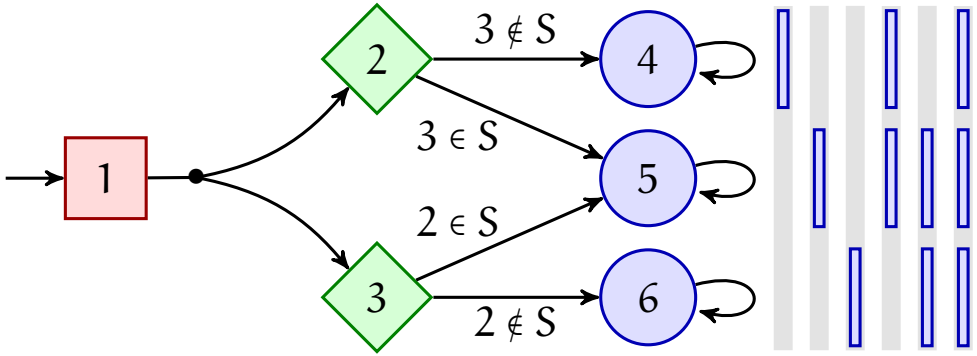
Alternating run



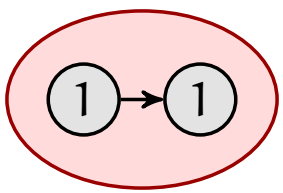
S: set of received states



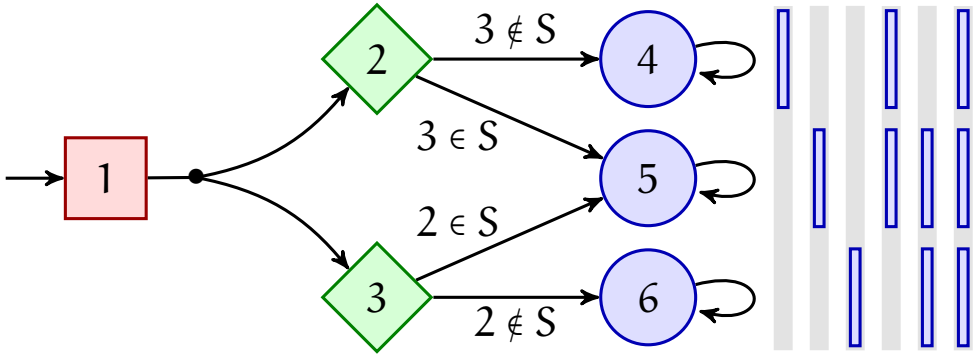
Alternating run



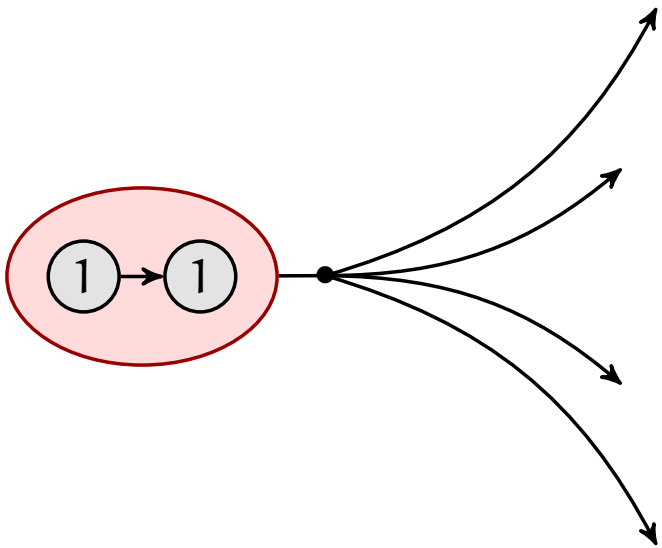
S: set of received states



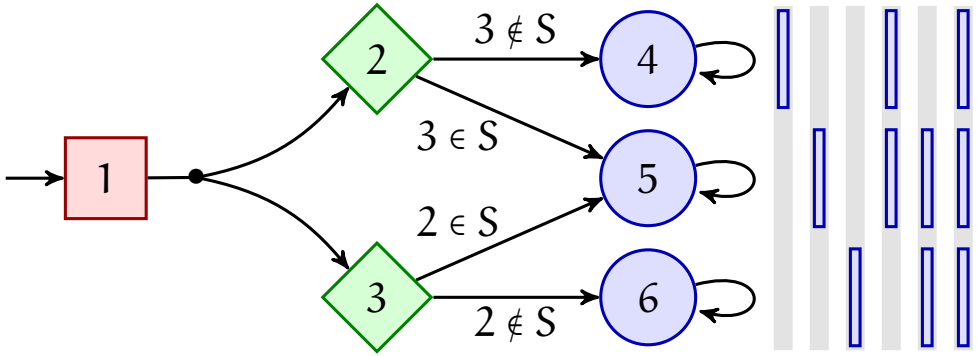
Alternating run



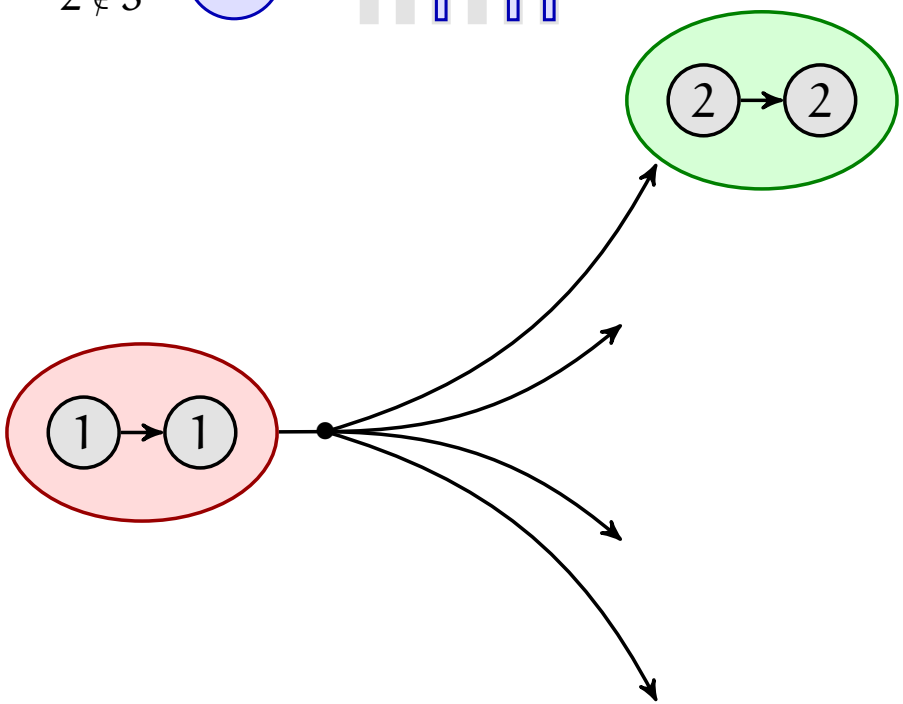
S: set of received states



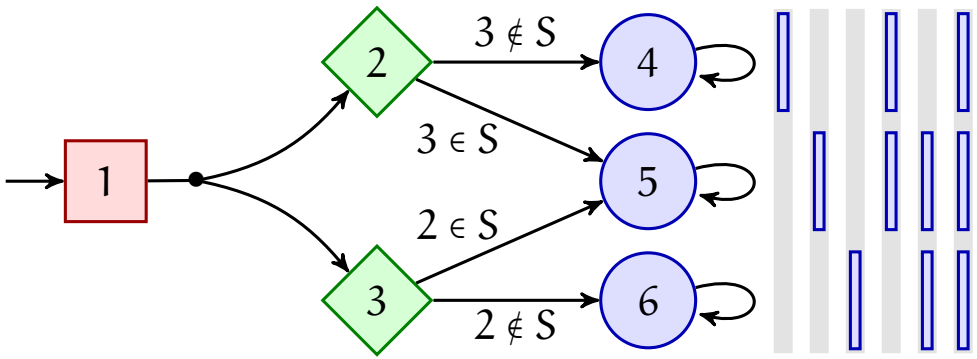
Alternating run



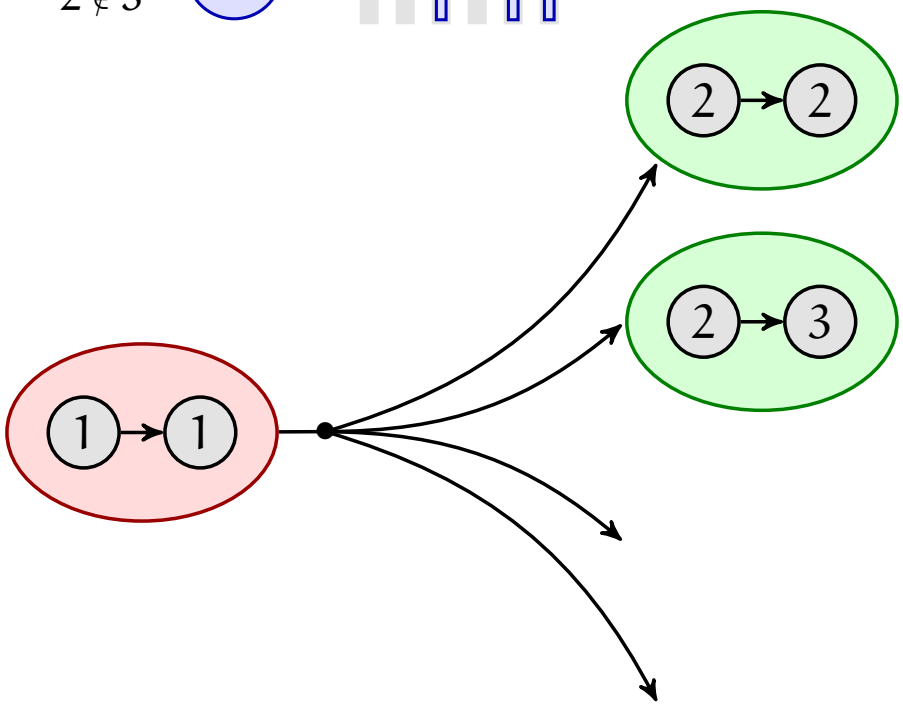
S: set of received states



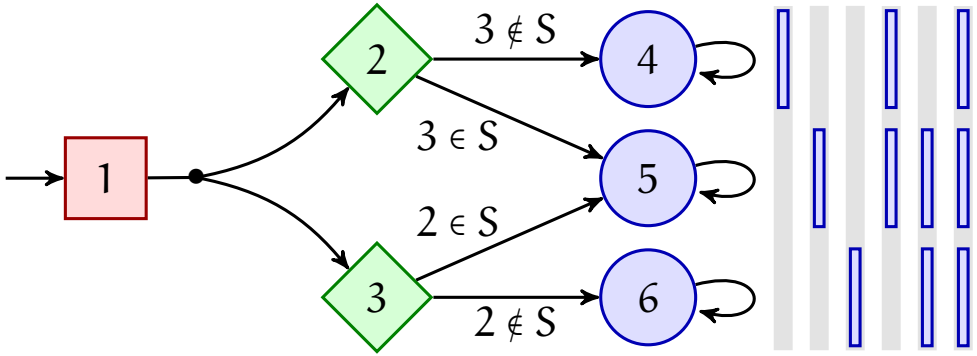
Alternating run



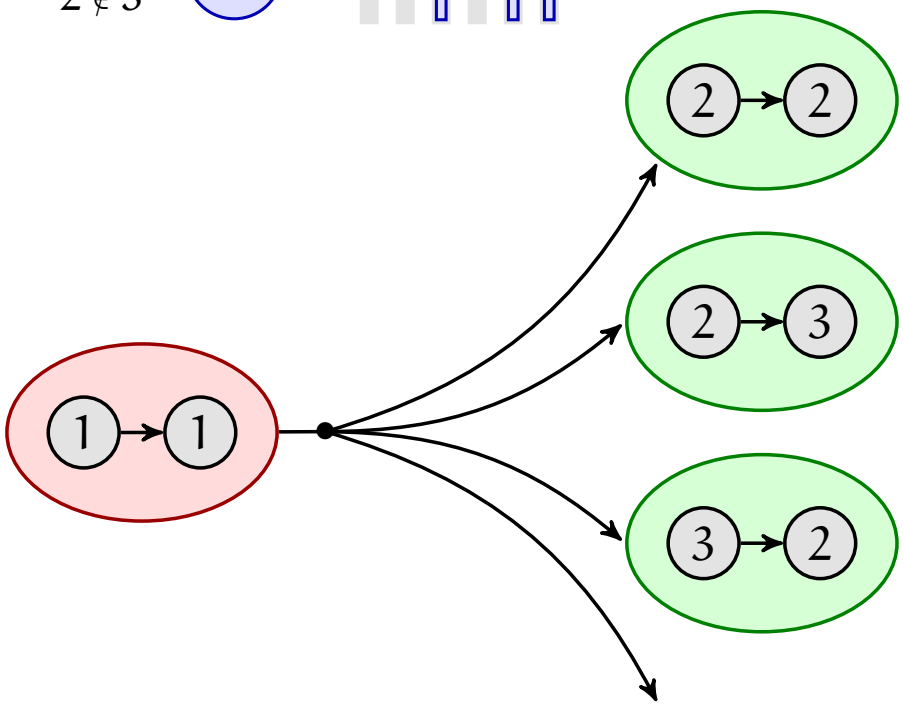
S: set of received states



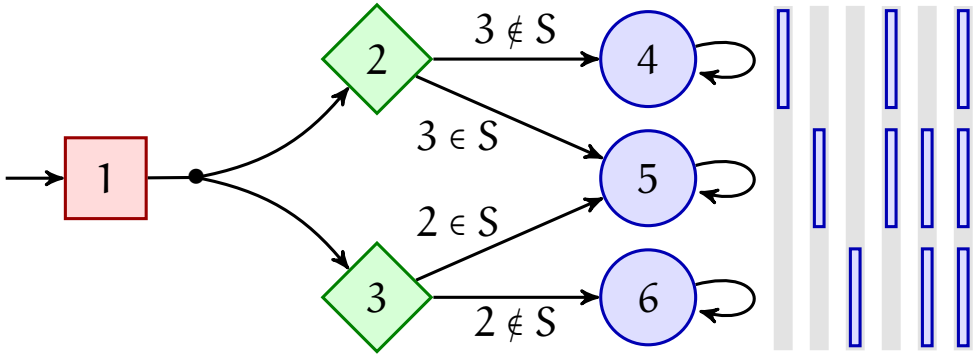
Alternating run



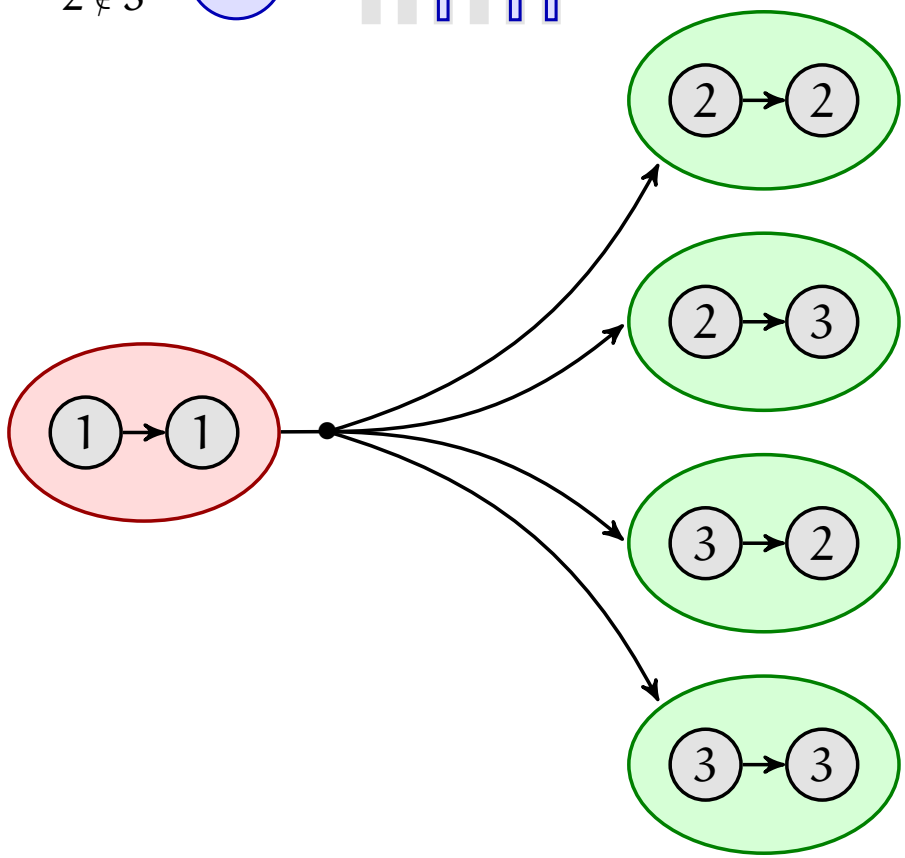
S: set of received states



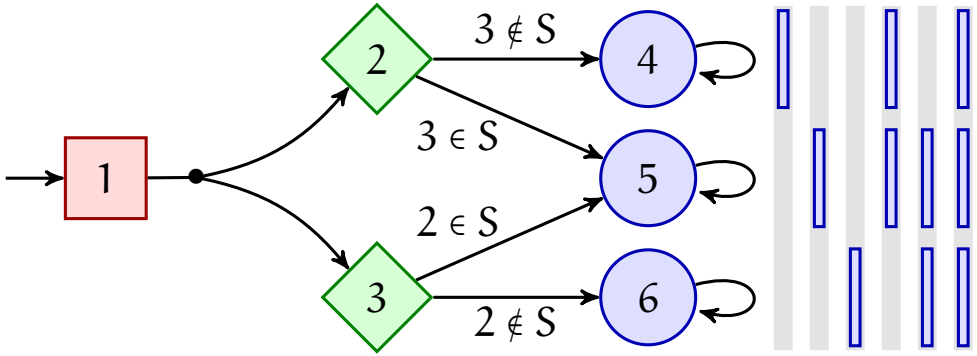
Alternating run



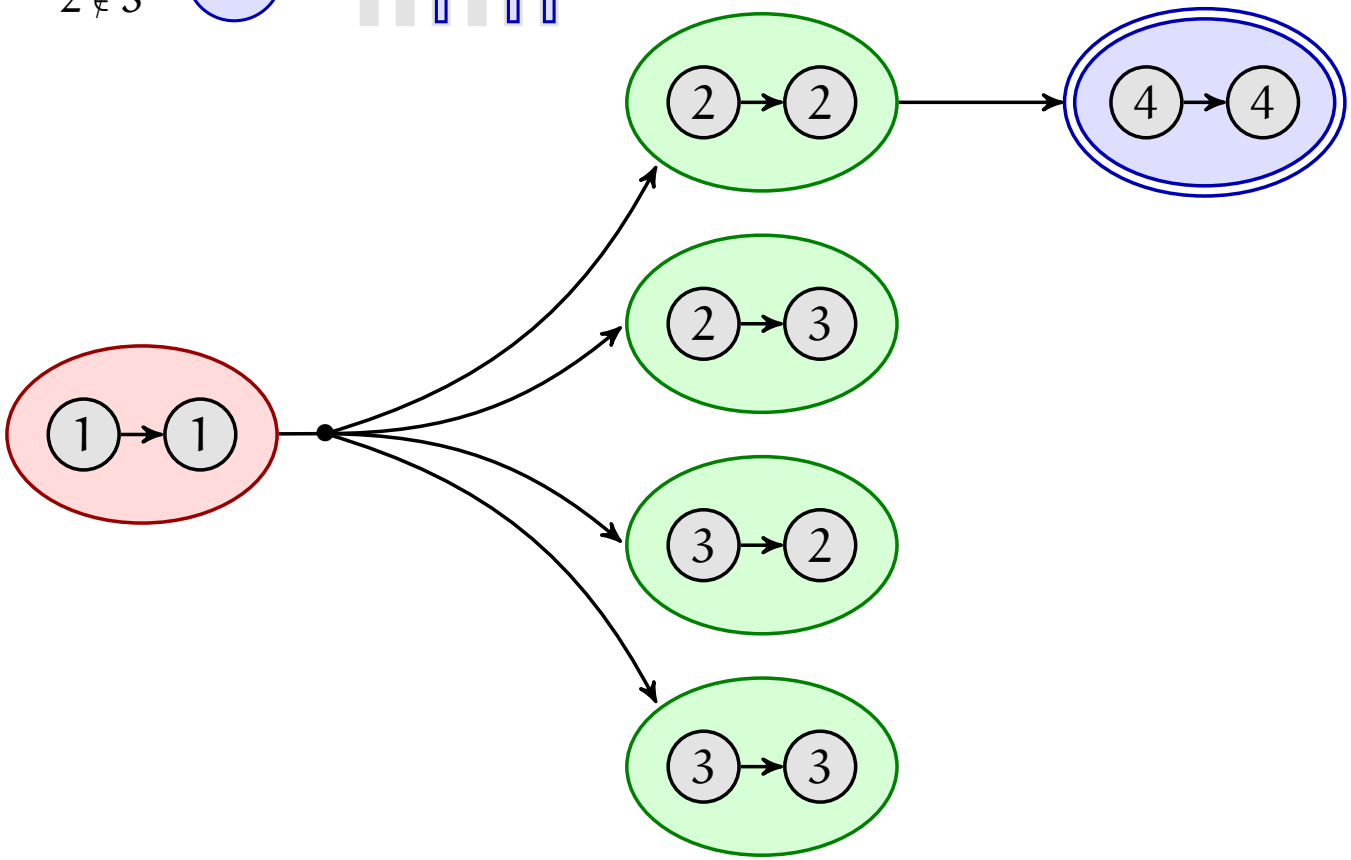
S: set of received states



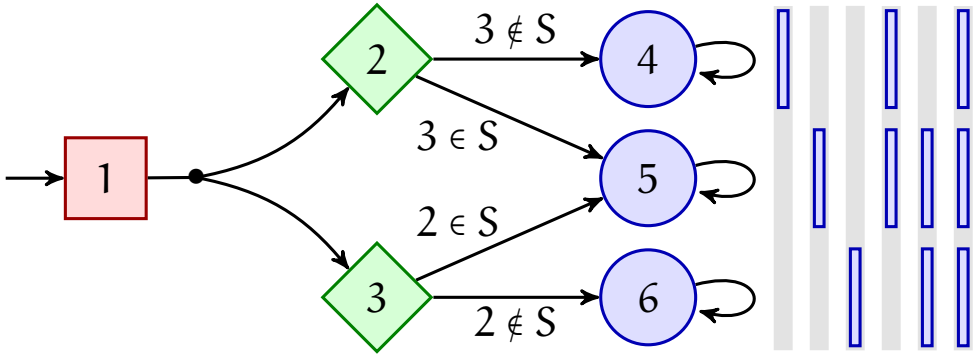
Alternating run



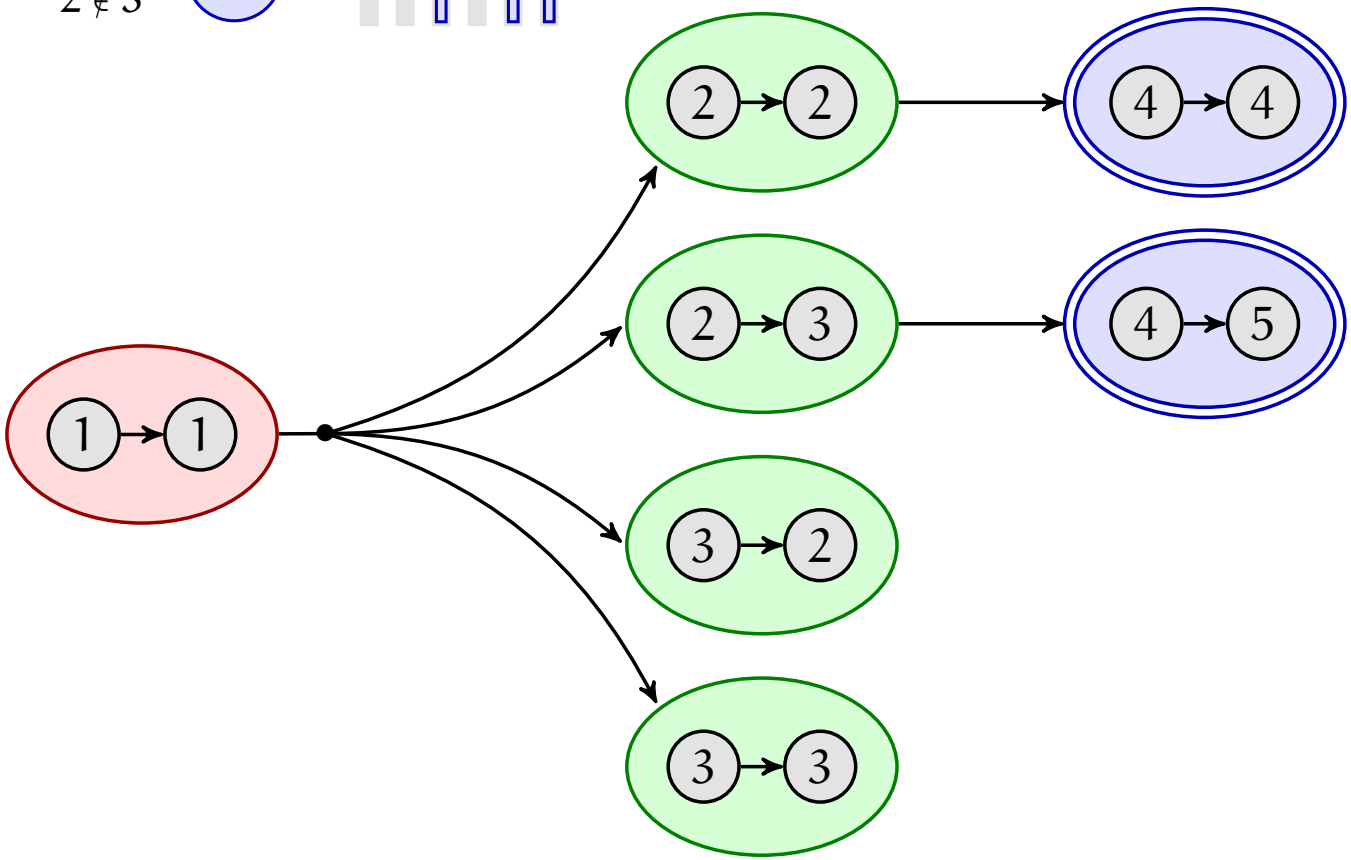
S: set of received states



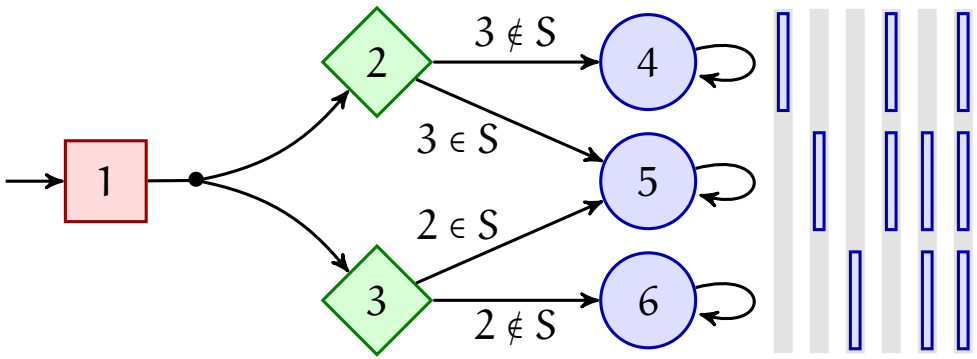
Alternating run



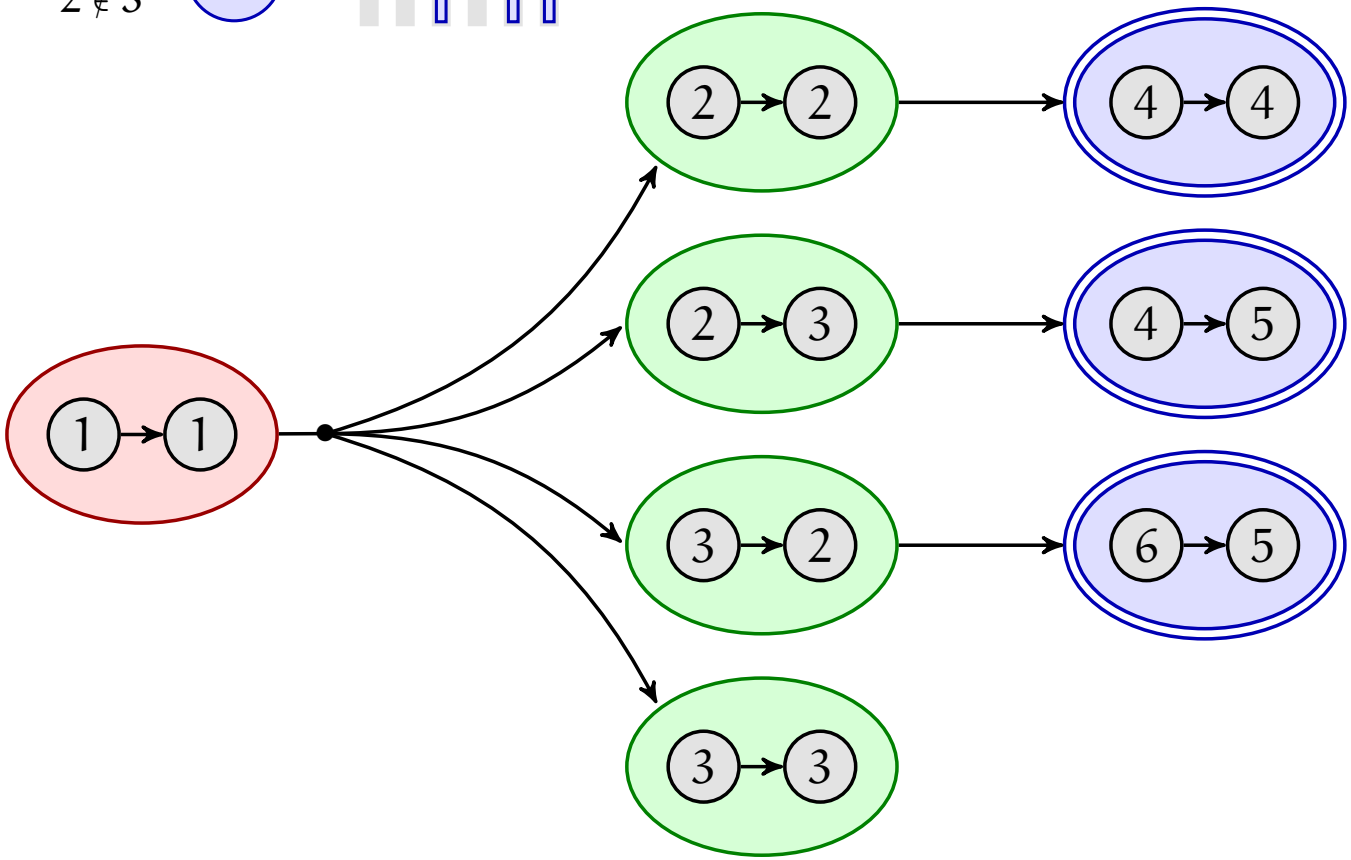
S: set of received states



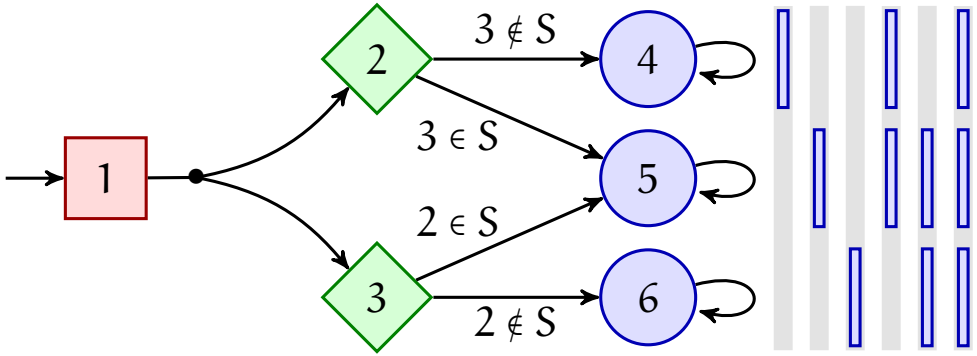
Alternating run



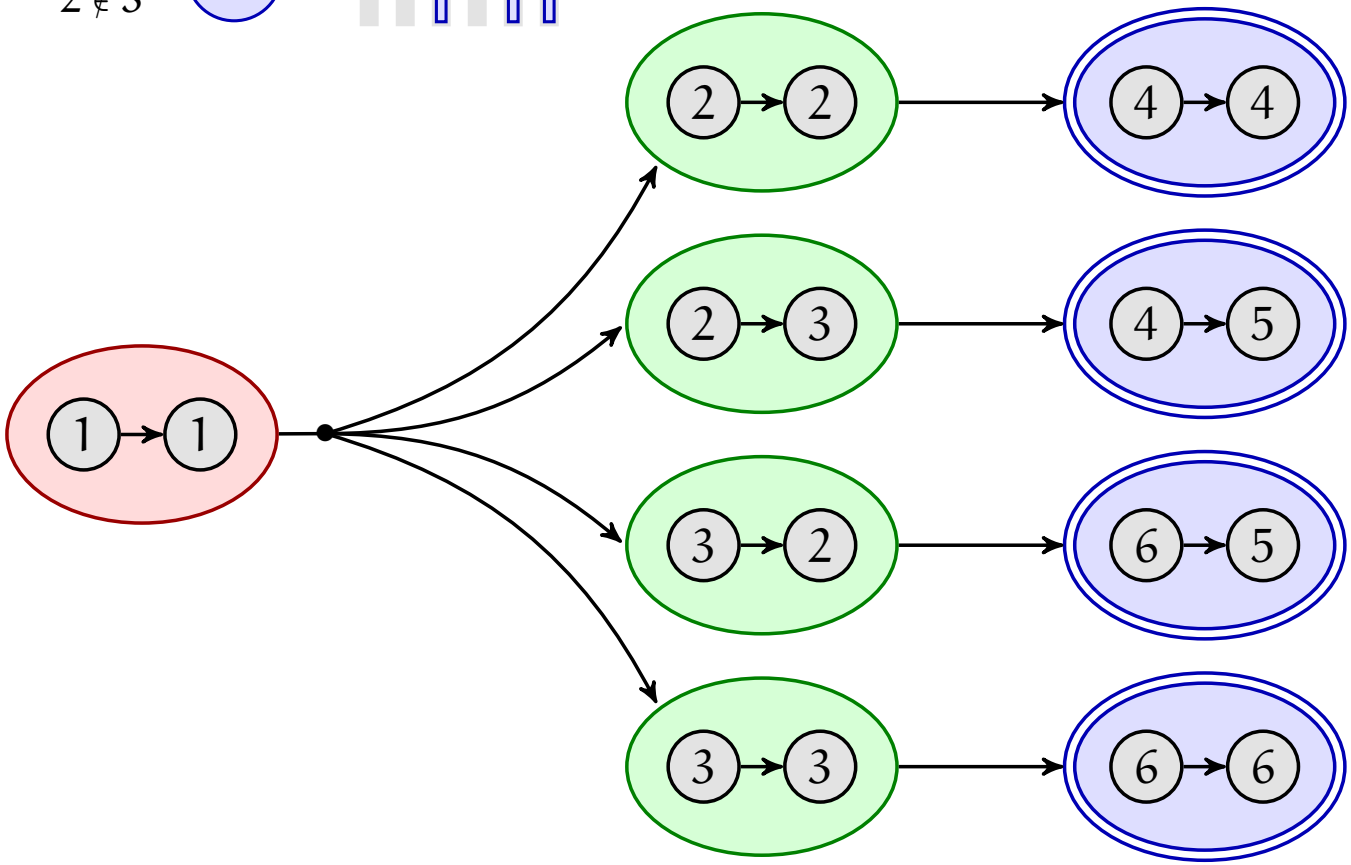
S: set of received states



Alternating run



S: set of received states



Contributions

MONADIC SECOND-ORDER LOGIC

ALTERNATING LOCAL AUTOMATA

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

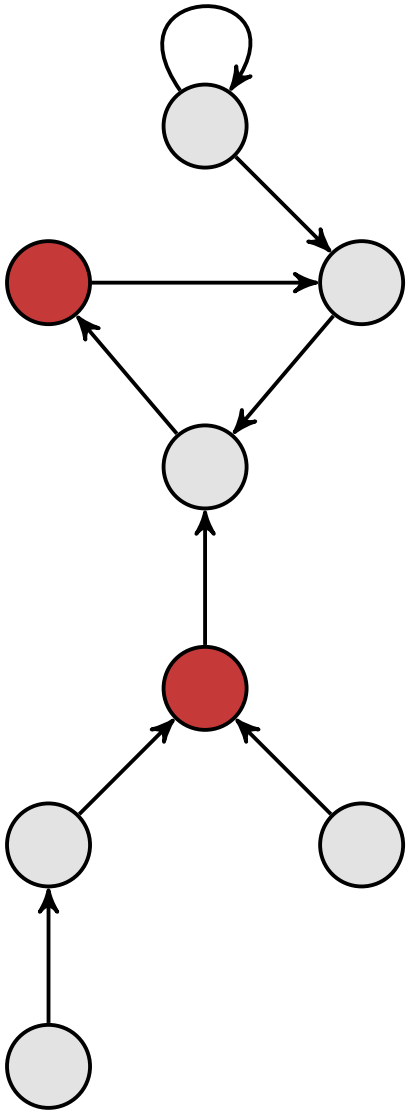
$$\delta \colon Q \times 2^Q \rightarrow Q$$

\+ Unbounded running time

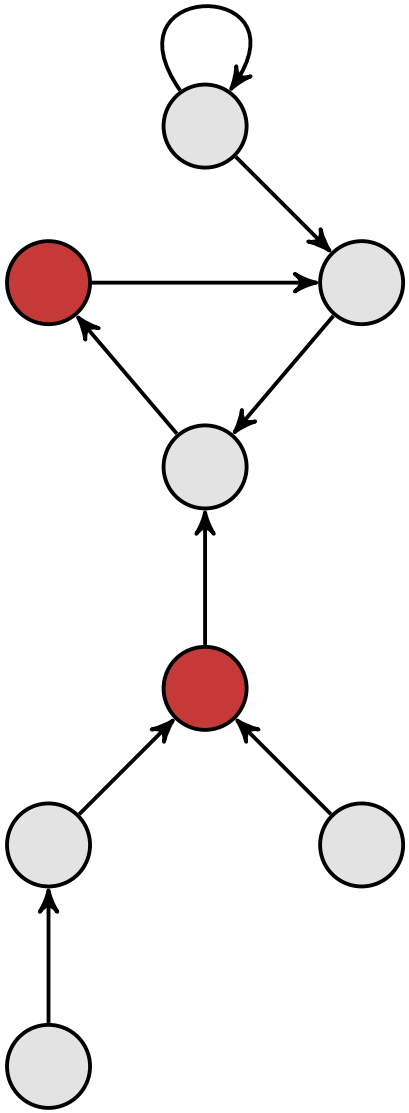
− Asynchronous execution

The backward μ-fragment

The backward μ-fragment

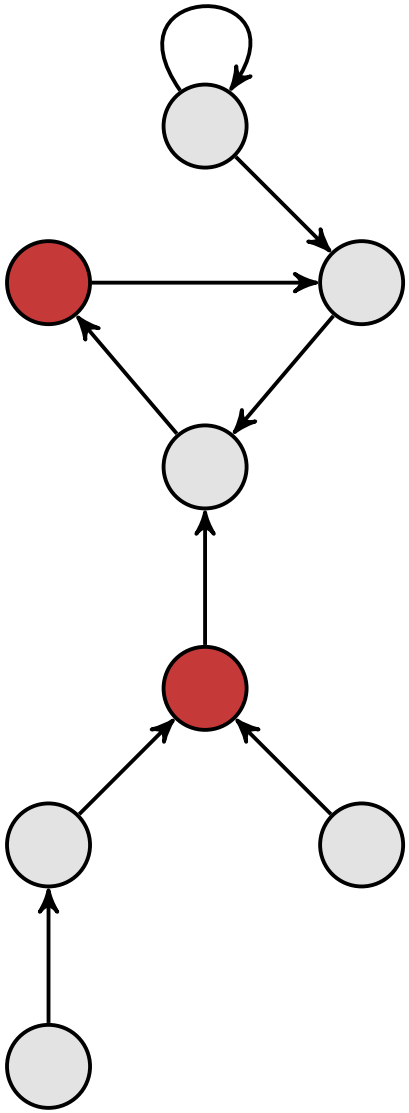


The backward μ-fragment



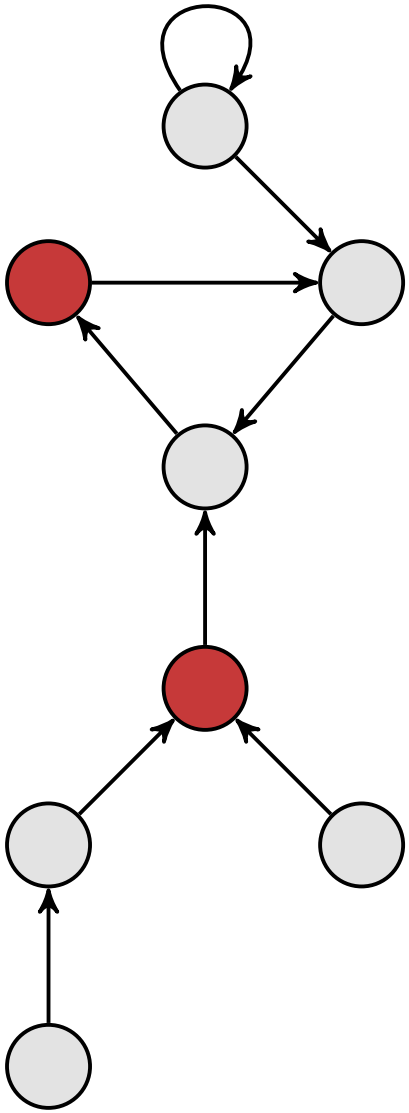
$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \Big(\qquad \Big)$$

The backward μ-fragment



$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \end{pmatrix}$$

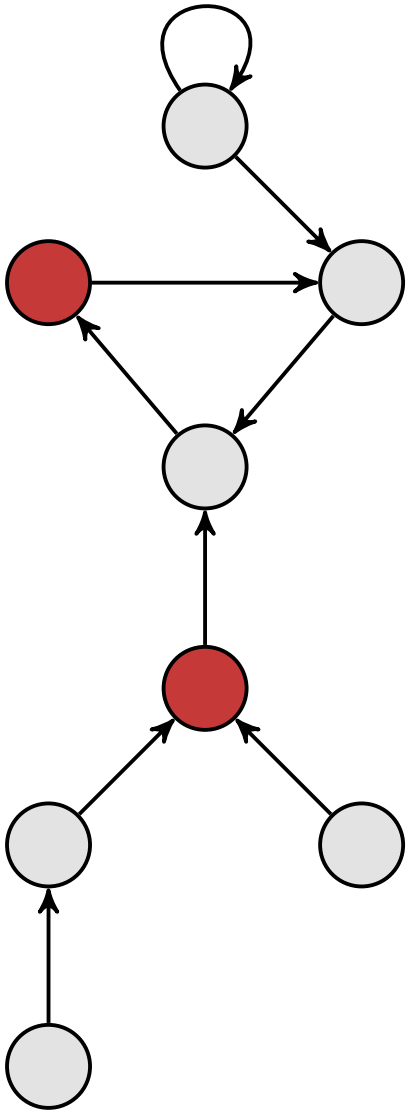
The backward μ-fragment



constant

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \end{pmatrix}$$

The backward μ-fragment

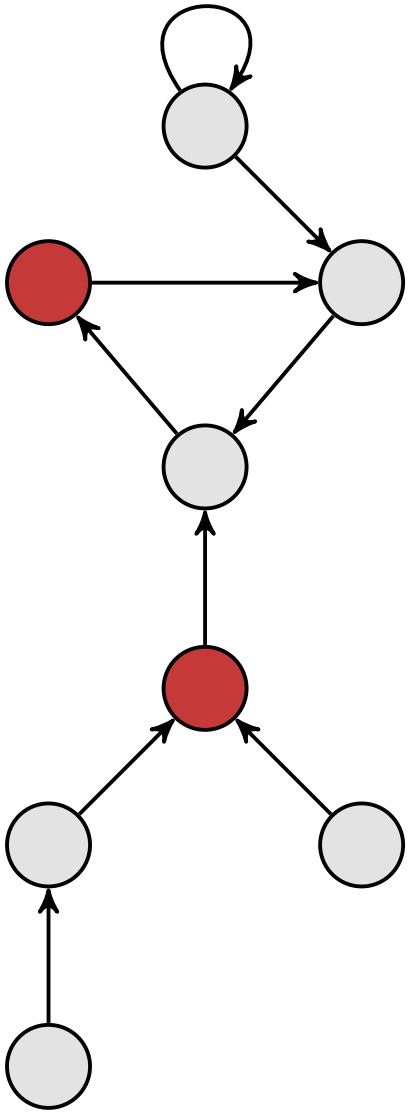


unnegated variable

constant

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \end{pmatrix}$$

The backward μ-fragment



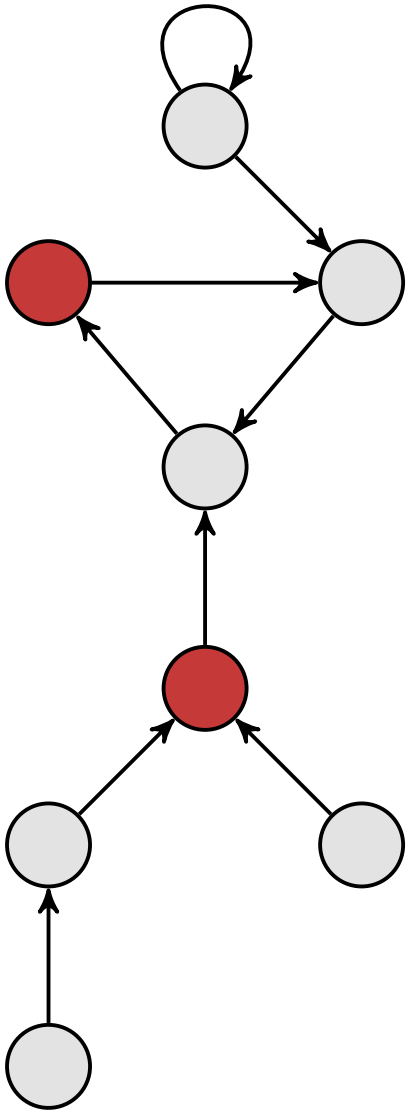
unnegated variable

constant

∃ incoming neighbor

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \left(\begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \end{pmatrix} \right)$$

The backward μ-fragment



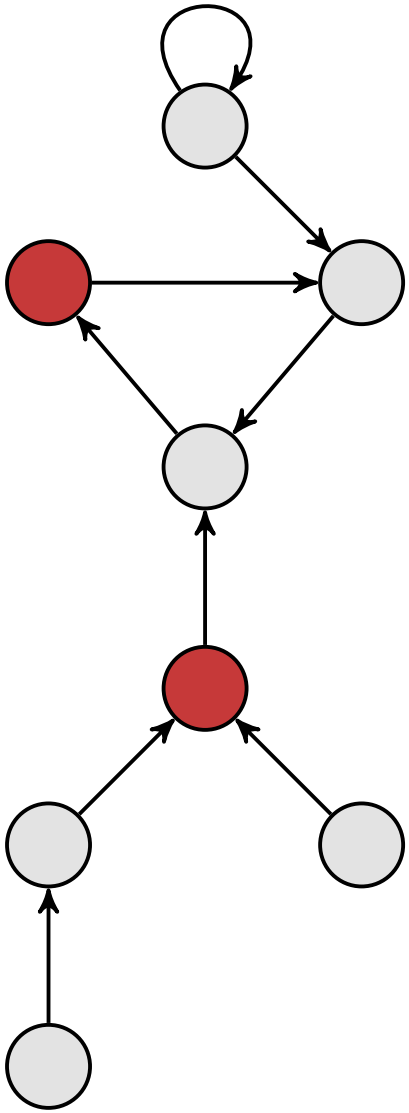
unnegated variable

constant

∃ incoming neighbor

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

The backward μ-fragment



unnegated variable

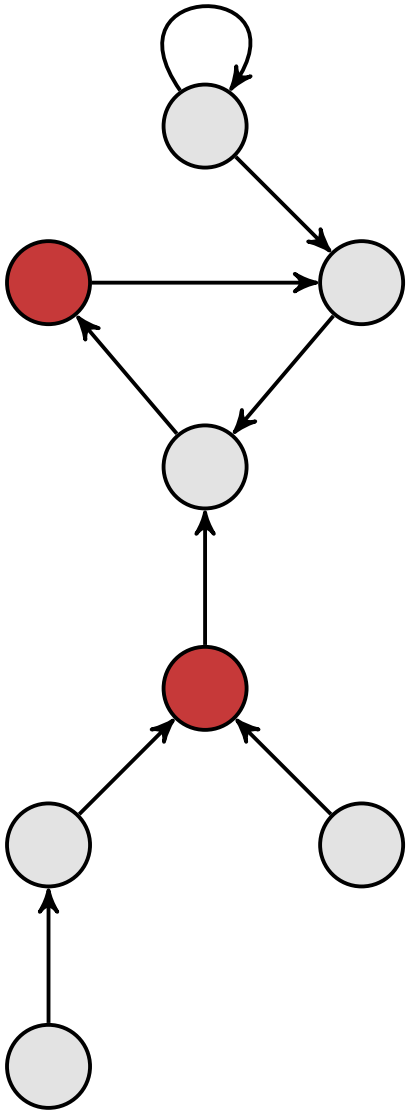
constant

∃ incoming neighbor

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

∀ incoming neighbors

The backward μ-fragment



unnegated variable

constant

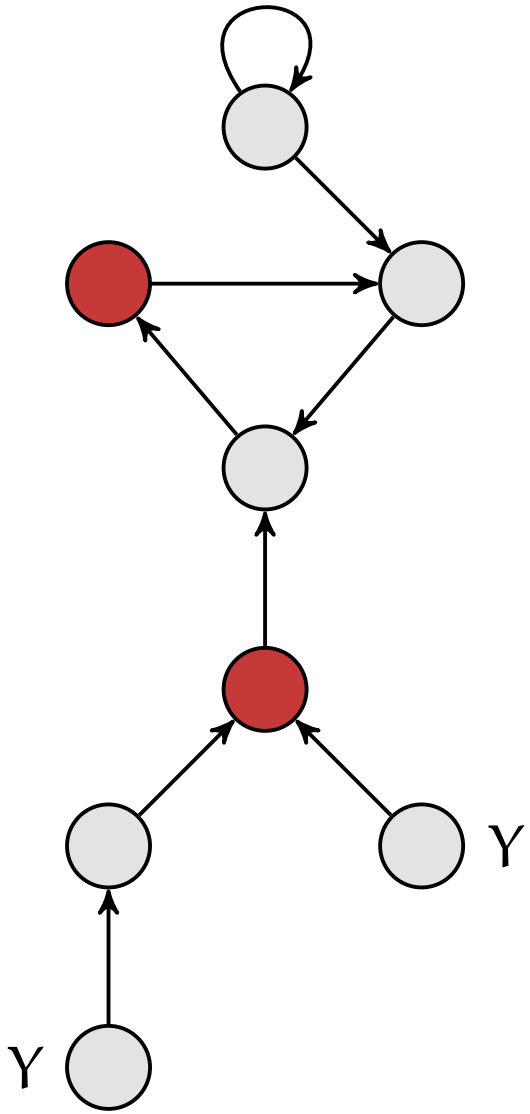
∃ incoming neighbor

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

∀ incoming neighbors

Compute the simultaneous least fixpoint.

The backward μ-fragment



unnegated variable

constant

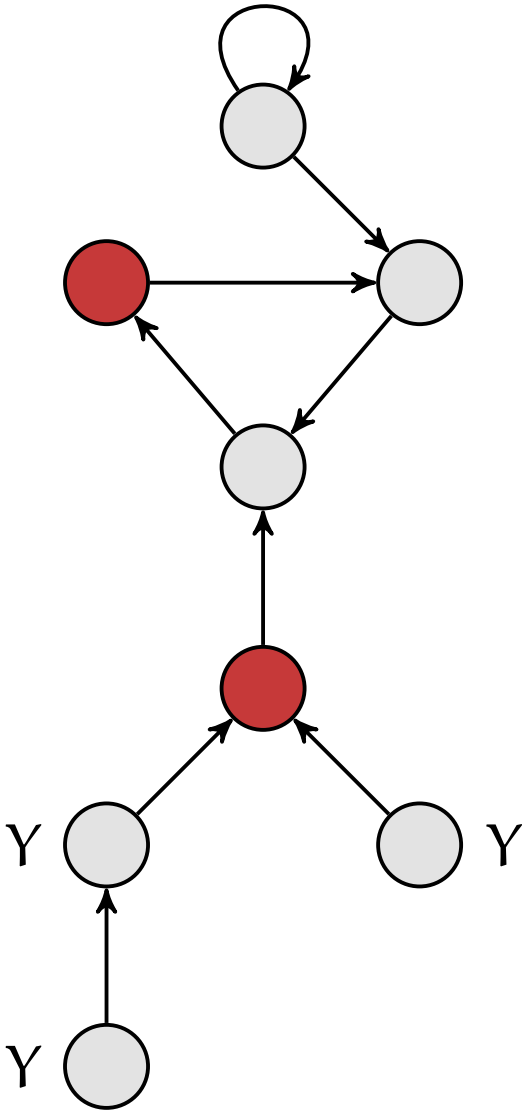
∃ incoming neighbor

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

∀ incoming neighbors

Compute the simultaneous least fixpoint.

The backward μ-fragment



unnegated variable

constant

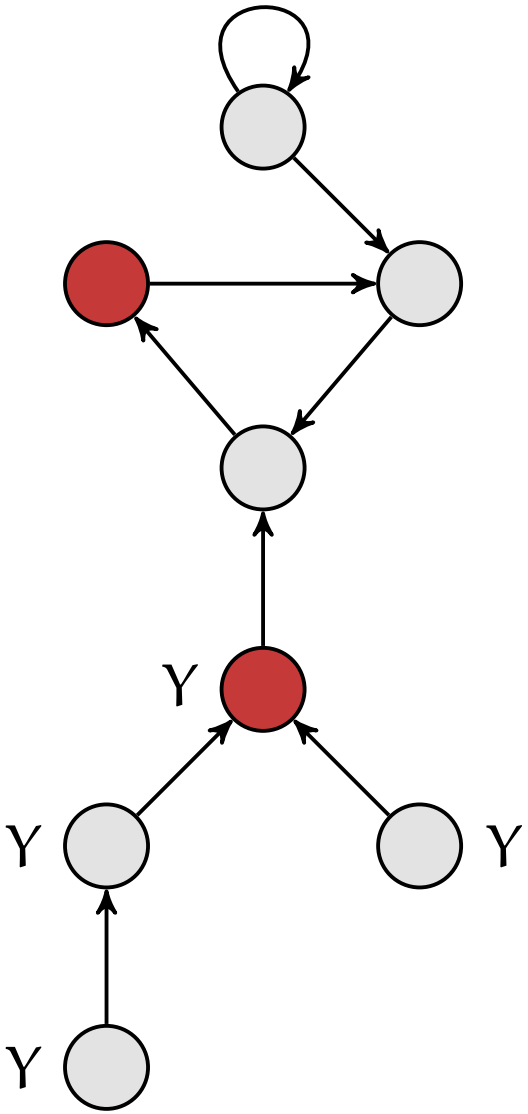
$\exists$ incoming neighbor

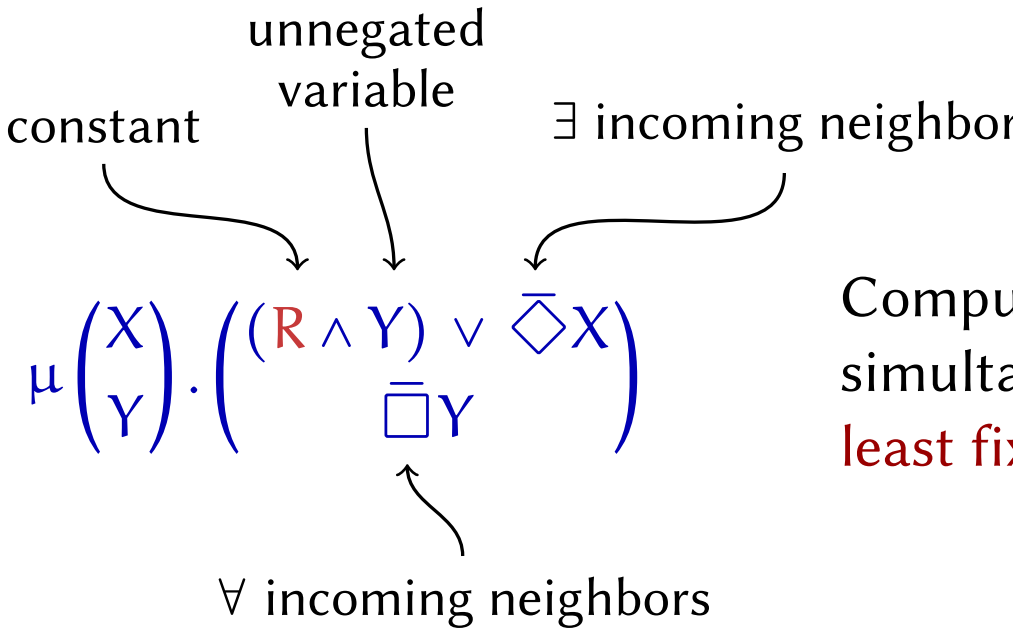
$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

$\forall$ incoming neighbors

Compute the simultaneous least fixpoint.

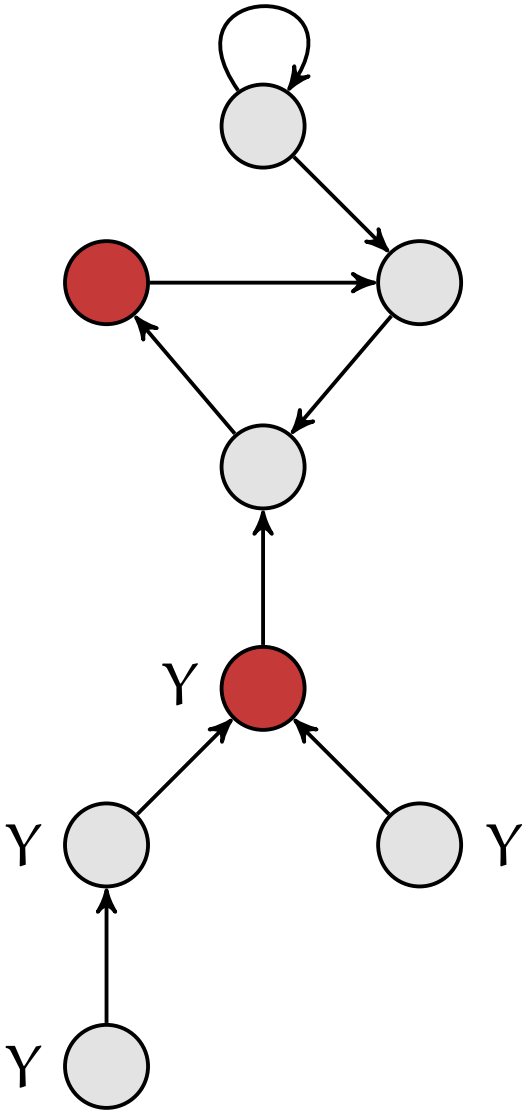
The backward μ-fragment

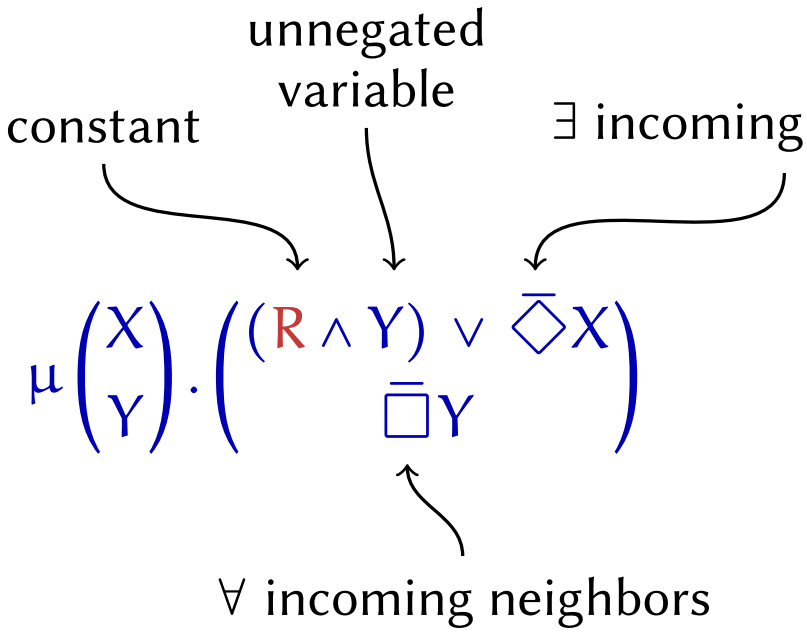




Compute the simultaneous least fixpoint.

The backward μ-fragment

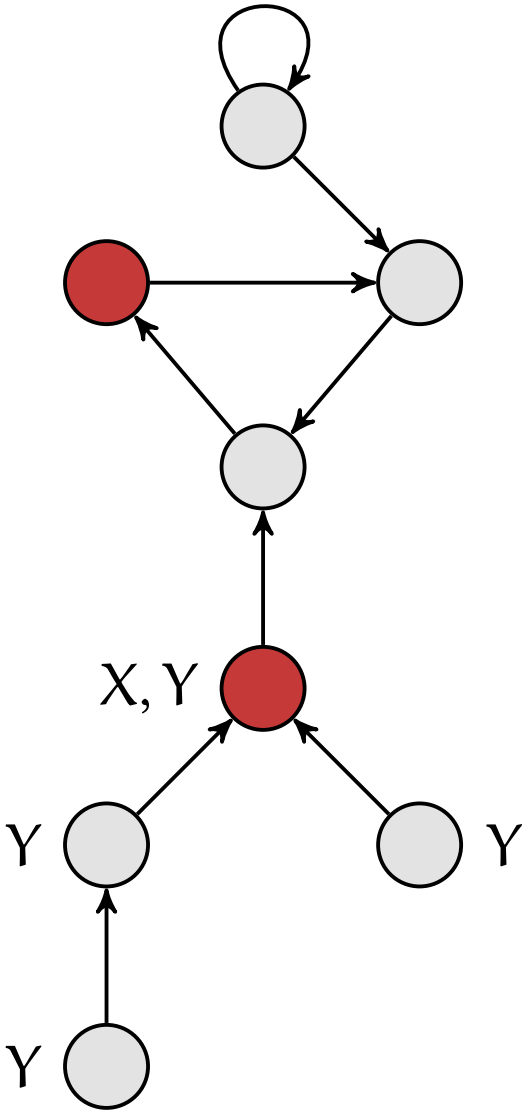




Compute the simultaneous least fixpoint.

Y: "Going backwards, we cannot reach any directed cycle (only dead-ends)."

The backward μ-fragment



unnegated variable

constant

∃ incoming neighbor

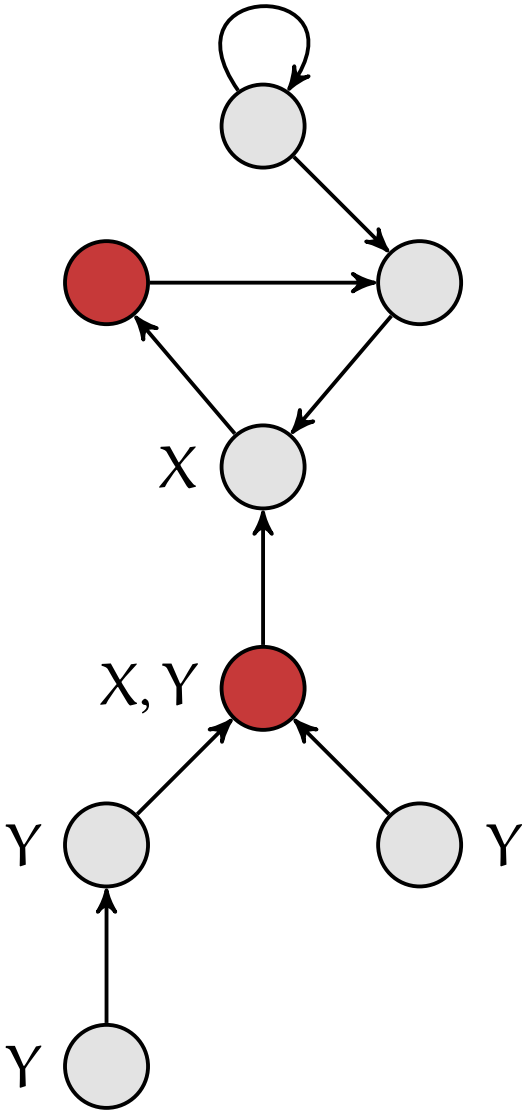
$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

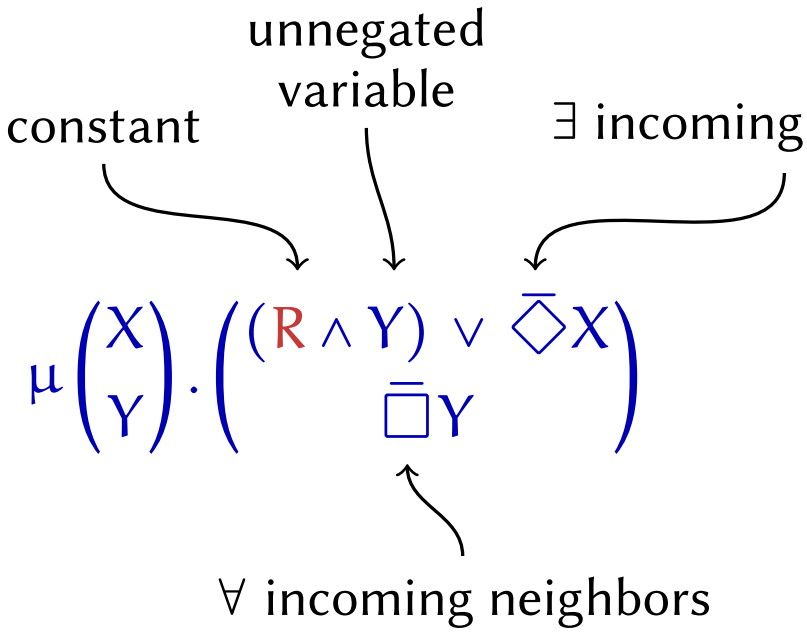
∀ incoming neighbors

Compute the simultaneous least fixpoint.

Y: "Going backwards, we cannot reach any directed cycle (only dead-ends)."

The backward μ-fragment

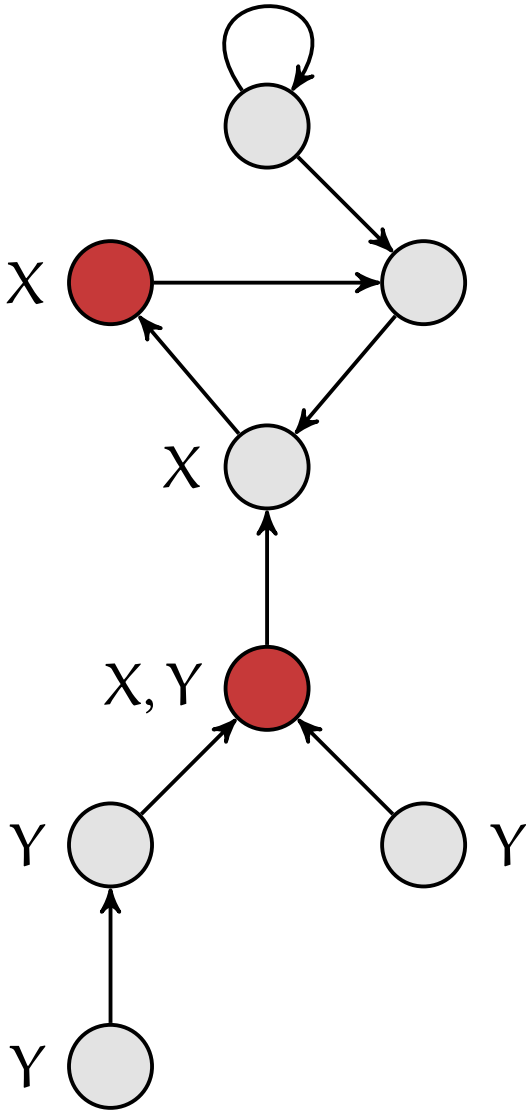




Compute the simultaneous least fixpoint.

Y: "Going backwards, we cannot reach any directed cycle (only dead-ends)."

The backward μ-fragment



unnegated variable

constant

∃ incoming neighbor

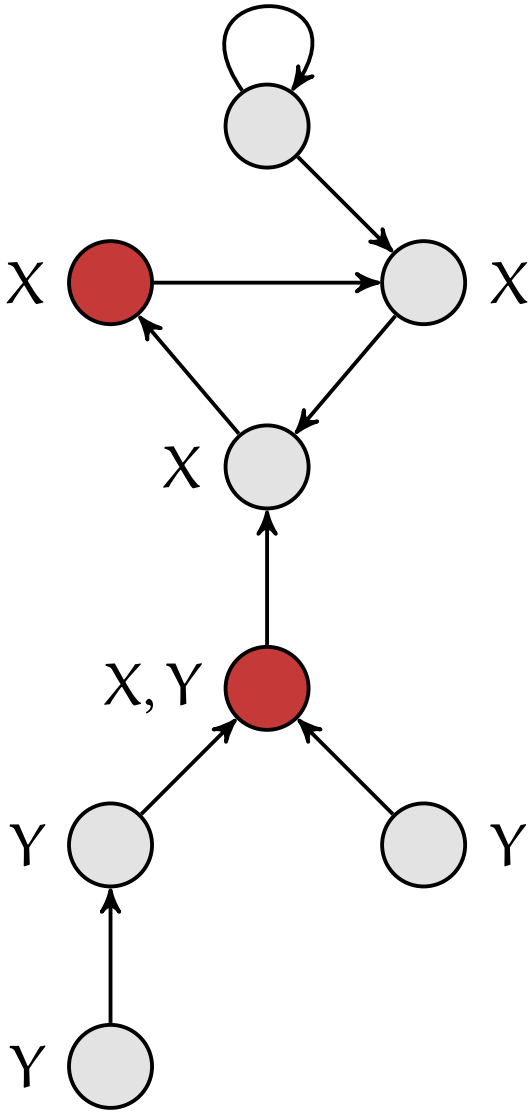
$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

∀ incoming neighbors

Compute the simultaneous least fixpoint.

Y: "Going backwards, we cannot reach any directed cycle (only dead-ends)."

The backward μ-fragment



unnegated variable

constant

∃ incoming neighbor

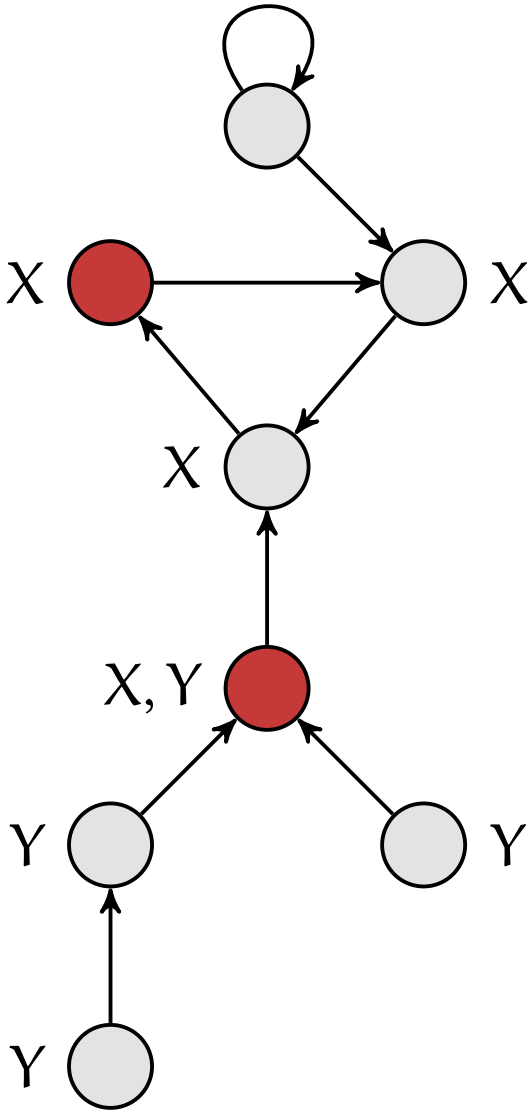
$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

∀ incoming neighbors

Compute the simultaneous least fixpoint.

Y: "Going backwards, we cannot reach any directed cycle (only dead-ends)."

The backward μ-fragment



unnegated variable

constant

∃ incoming neighbor

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \bar{\Diamond} X \\ \bar{\Box} Y \end{pmatrix}$$

∀ incoming neighbors

Compute the simultaneous least fixpoint.

Y: "Going backwards, we cannot reach any directed cycle (only dead-ends)."

X: "Going backwards, we can reach a red node from which no directed cycle is reachable."

Contributions

MONADIC SECOND-ORDER LOGIC

$$\forall Z \left(\exists x, y \big(Z(x) \wedge \neg Z(y) \big) \rightarrow \cdots \right)$$

EQUIVALENT

ALTERNATING LOCAL AUTOMATA

$$\delta \colon Q \times 2^Q \rightarrow 2^Q$$

\+ Alternation

\+ Global acceptance

THE BACKWARD μ-FRAGMENT

$$\mu \begin{pmatrix} X \\ Y \end{pmatrix} . \begin{pmatrix} (R \wedge Y) \vee \overline{\Diamond} X \\ \overline{\Box} Y \end{pmatrix}$$

EQUIVALENT

ASYNCHRONOUS AUTOMATA

with quasi-acyclic diagrams

$$\delta \colon Q \times 2^Q \rightarrow Q$$

\+ Unbounded running time

− Asynchronous execution

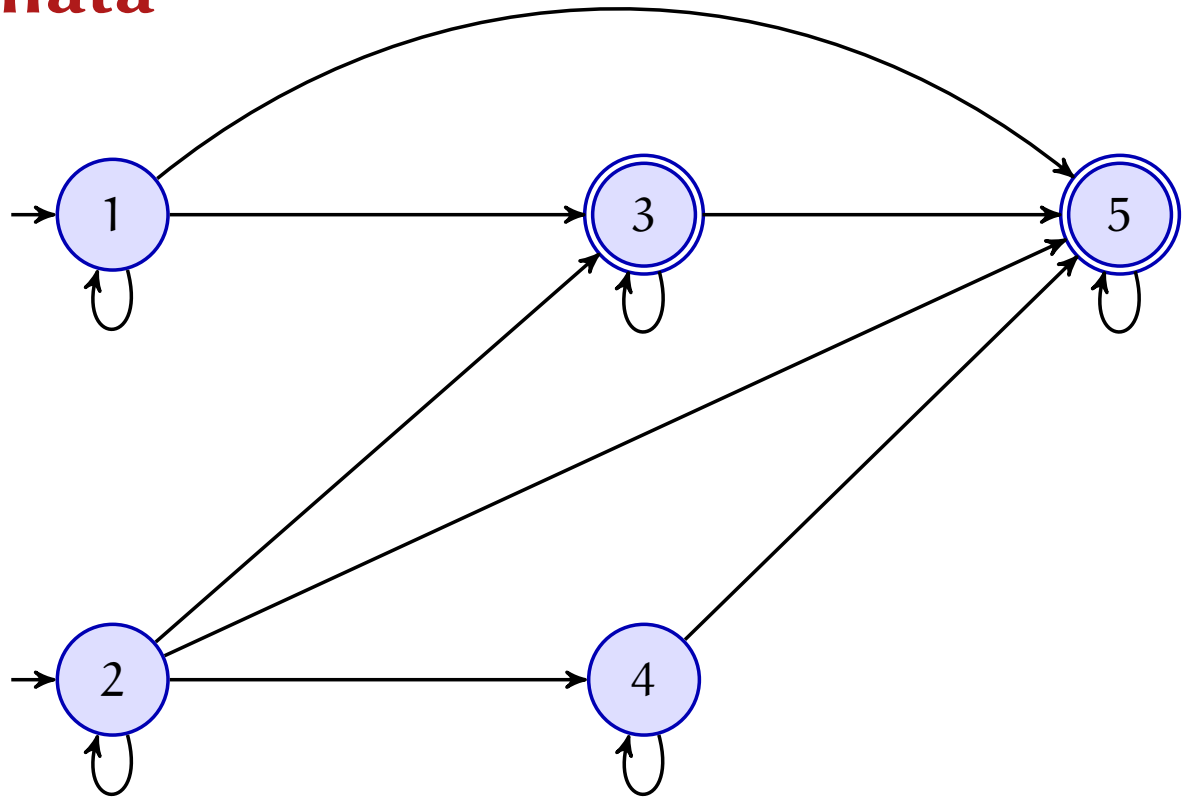
Asynchronous automata

Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Asynchronous automata

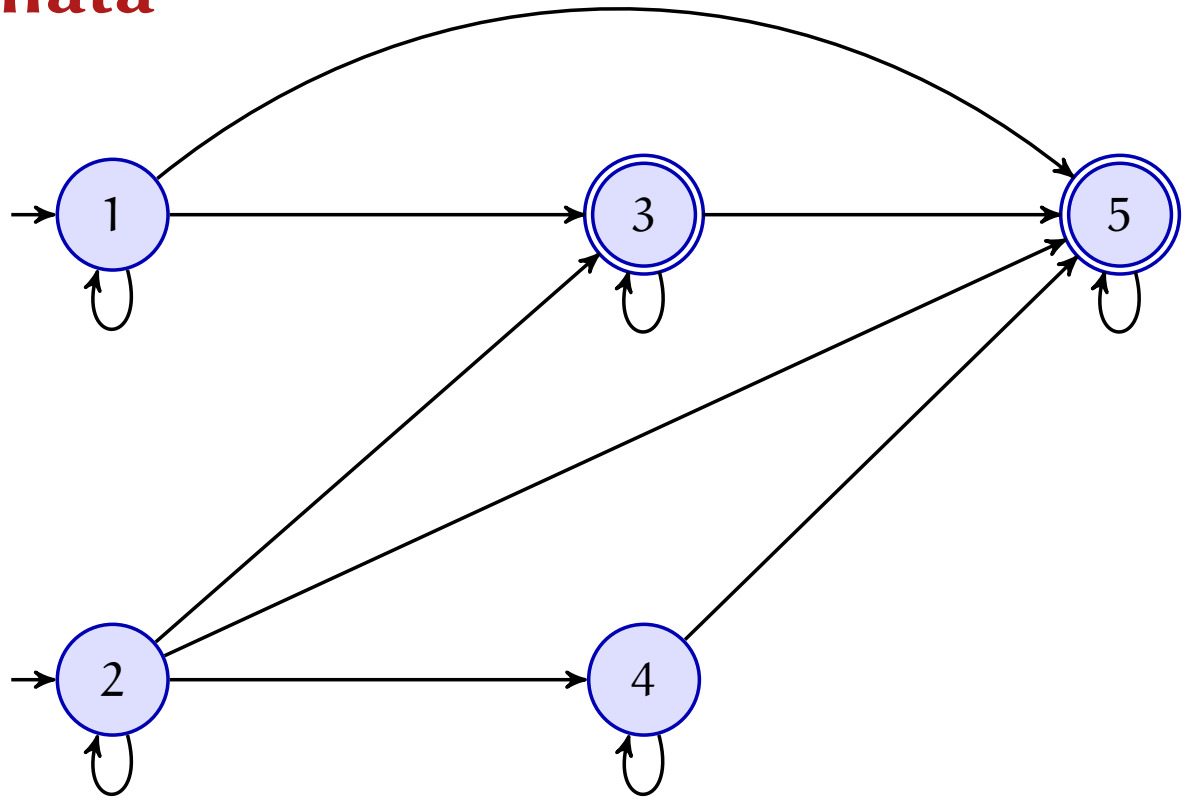
$\delta: Q \times 2^Q \to Q$



Asynchronous automata

$\delta: Q \times 2^Q \to Q$

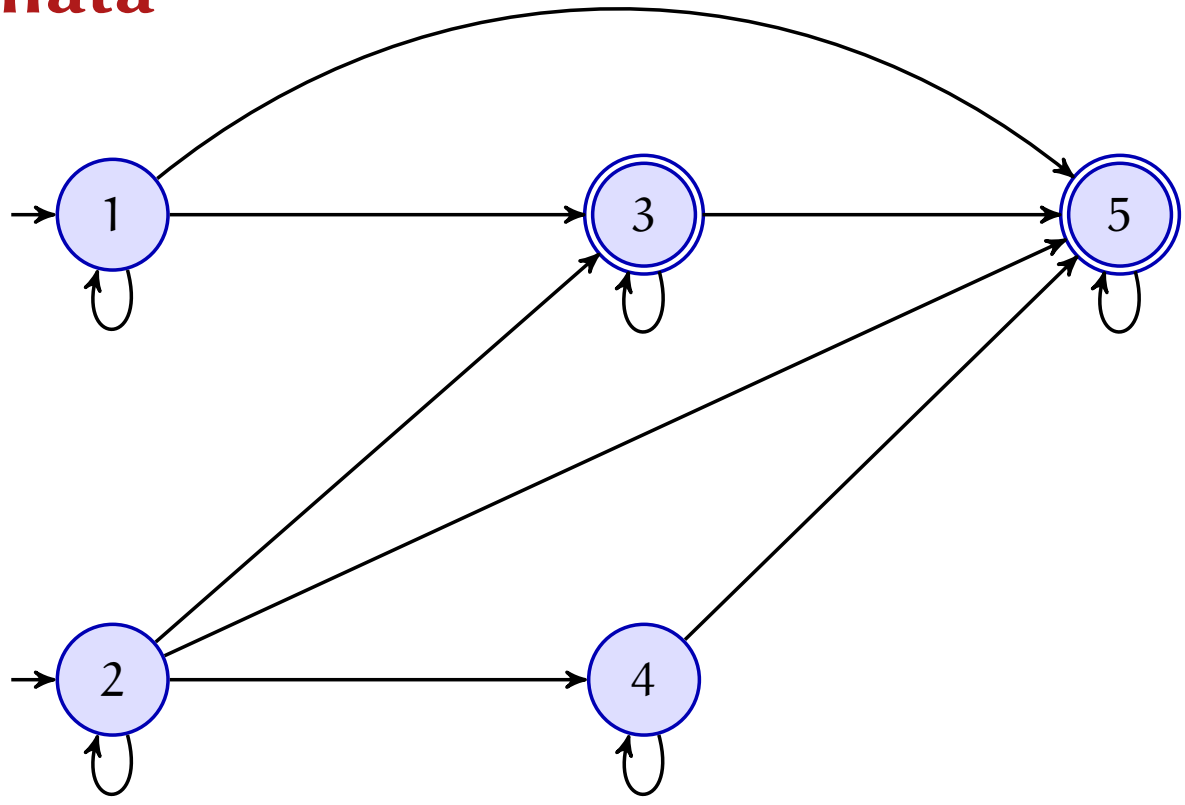
Quasi-acyclic
diagram.

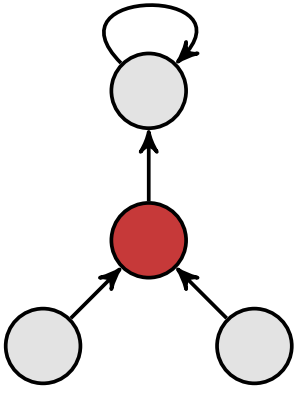


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic
diagram.

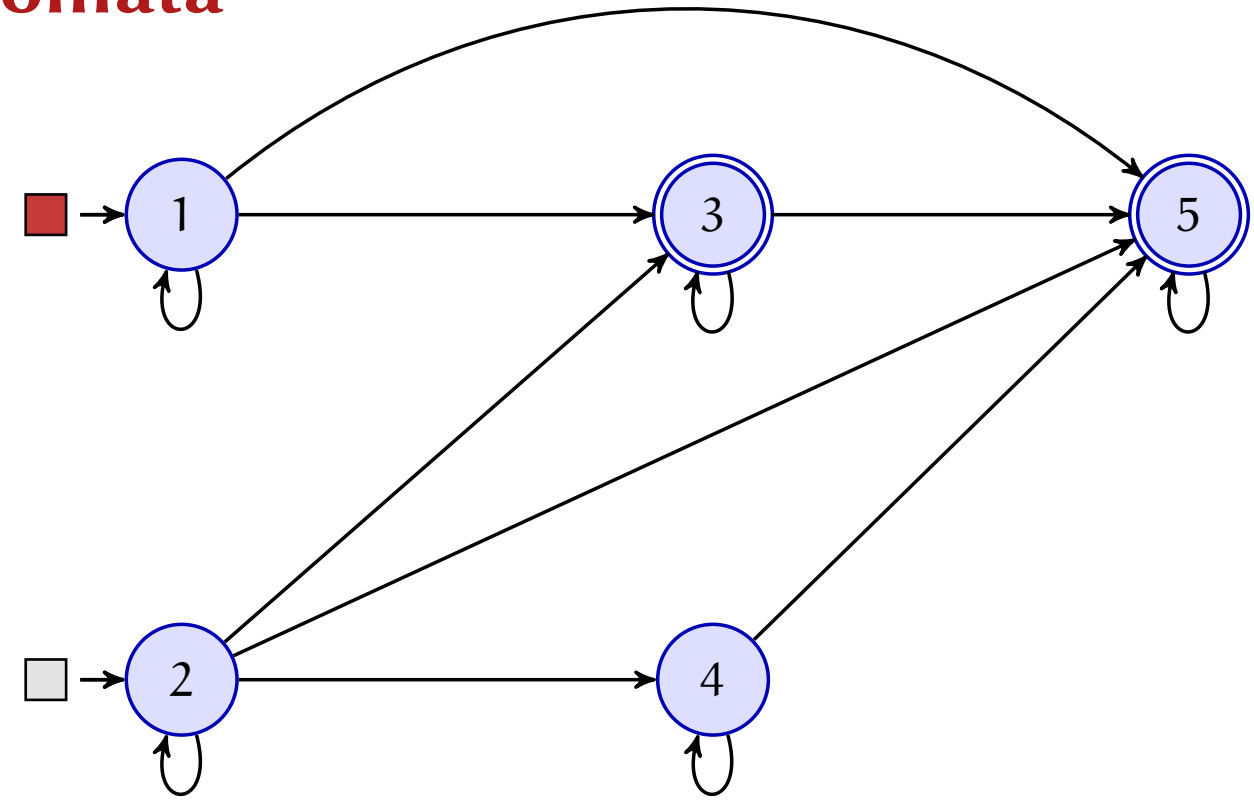


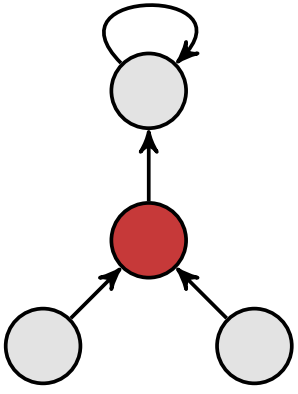


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic
diagram.

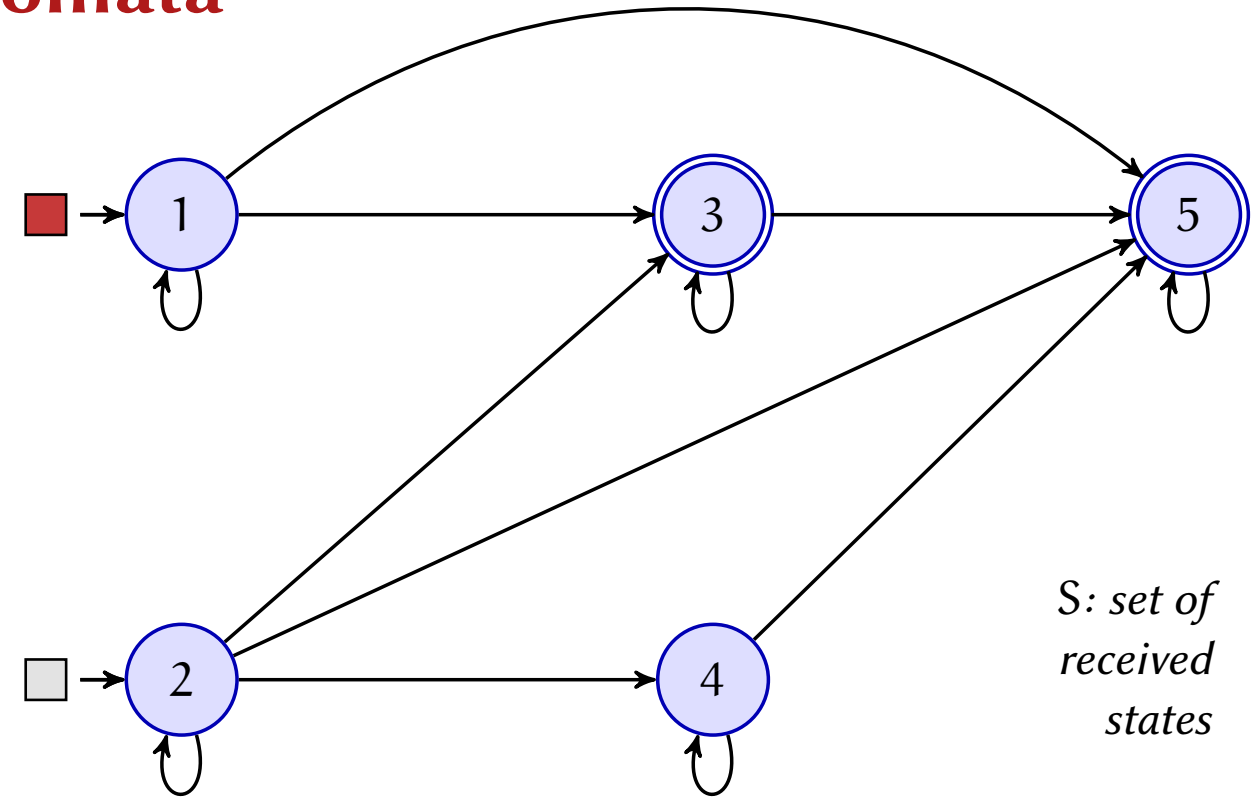




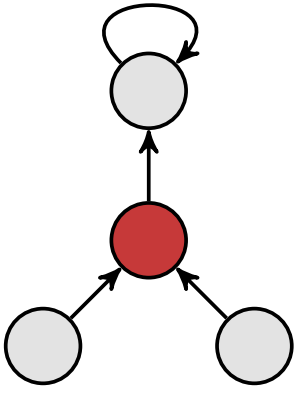
Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.



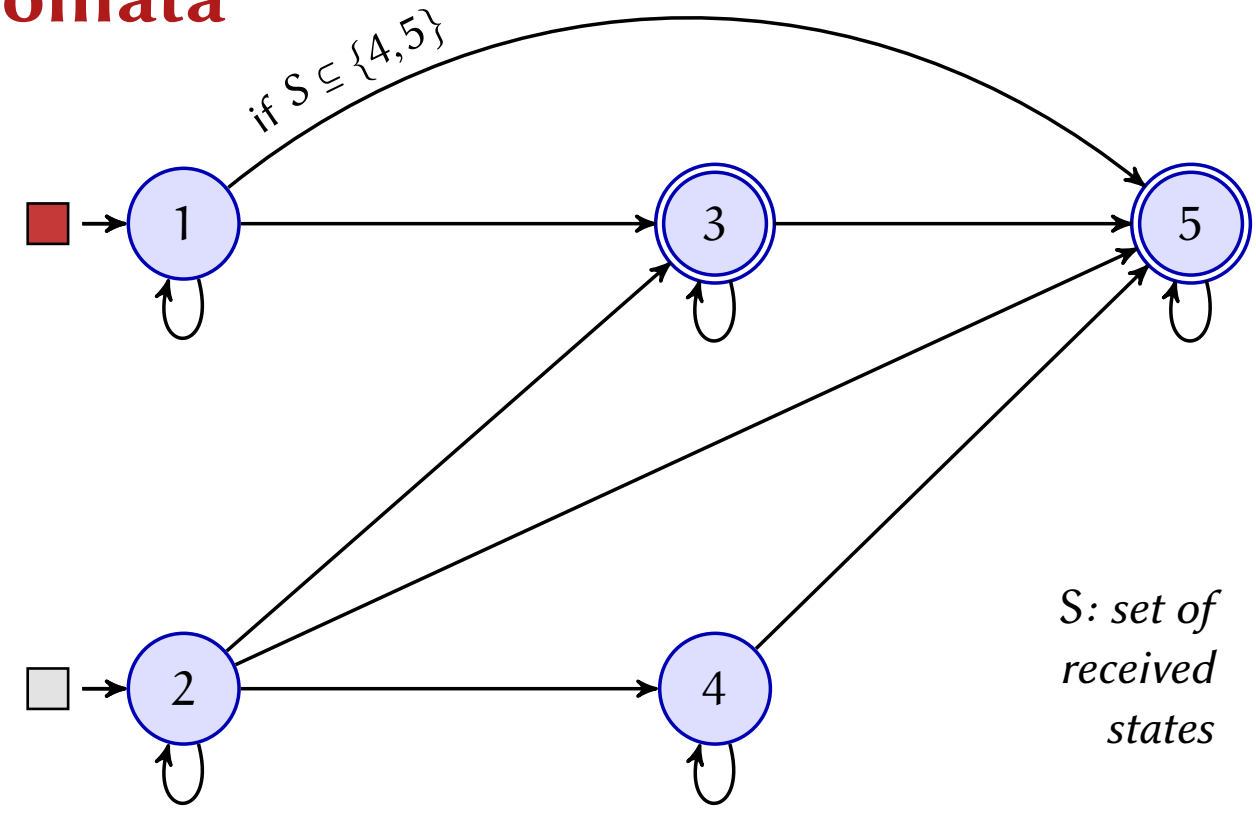
S: set of received states

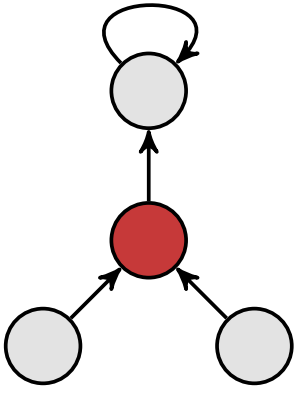


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

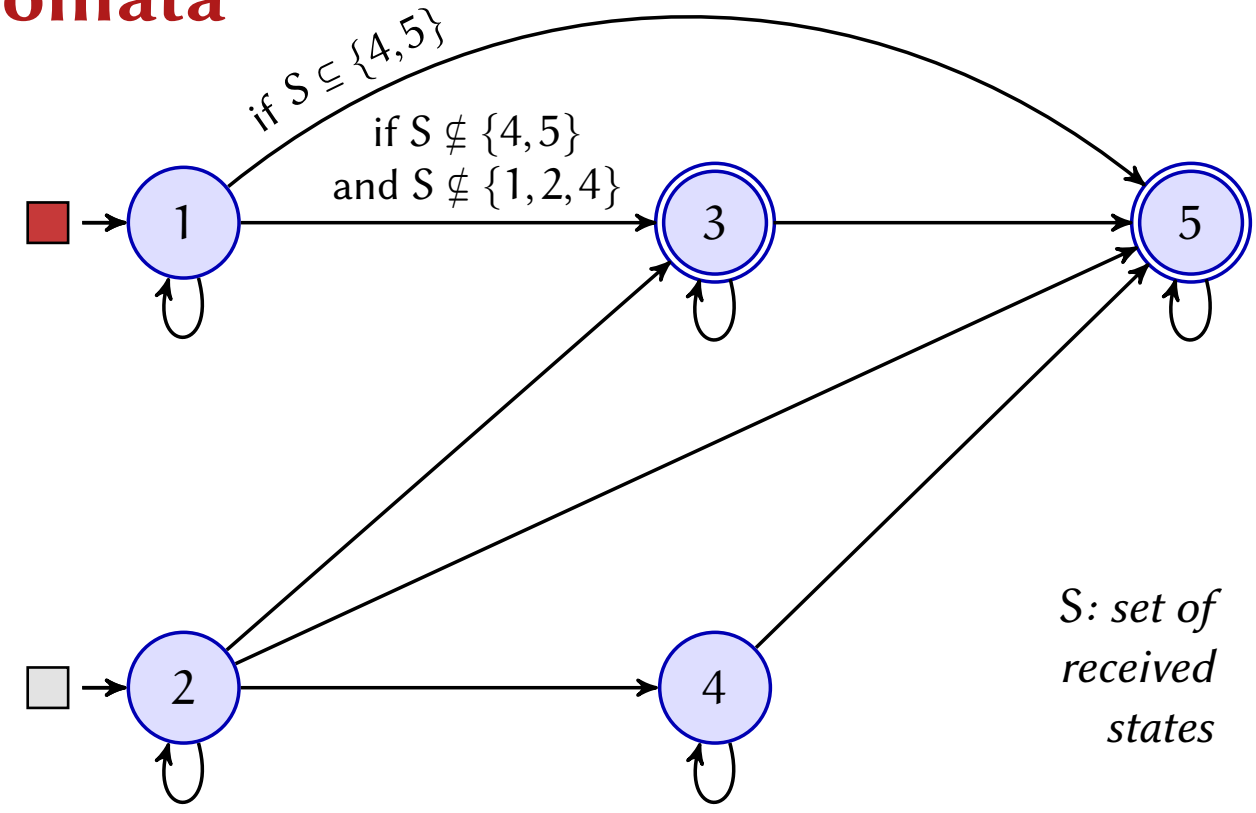


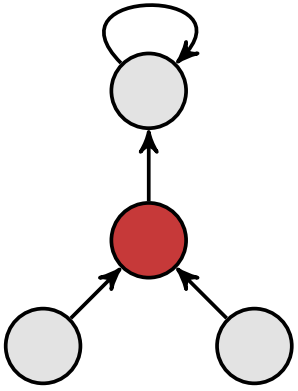


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

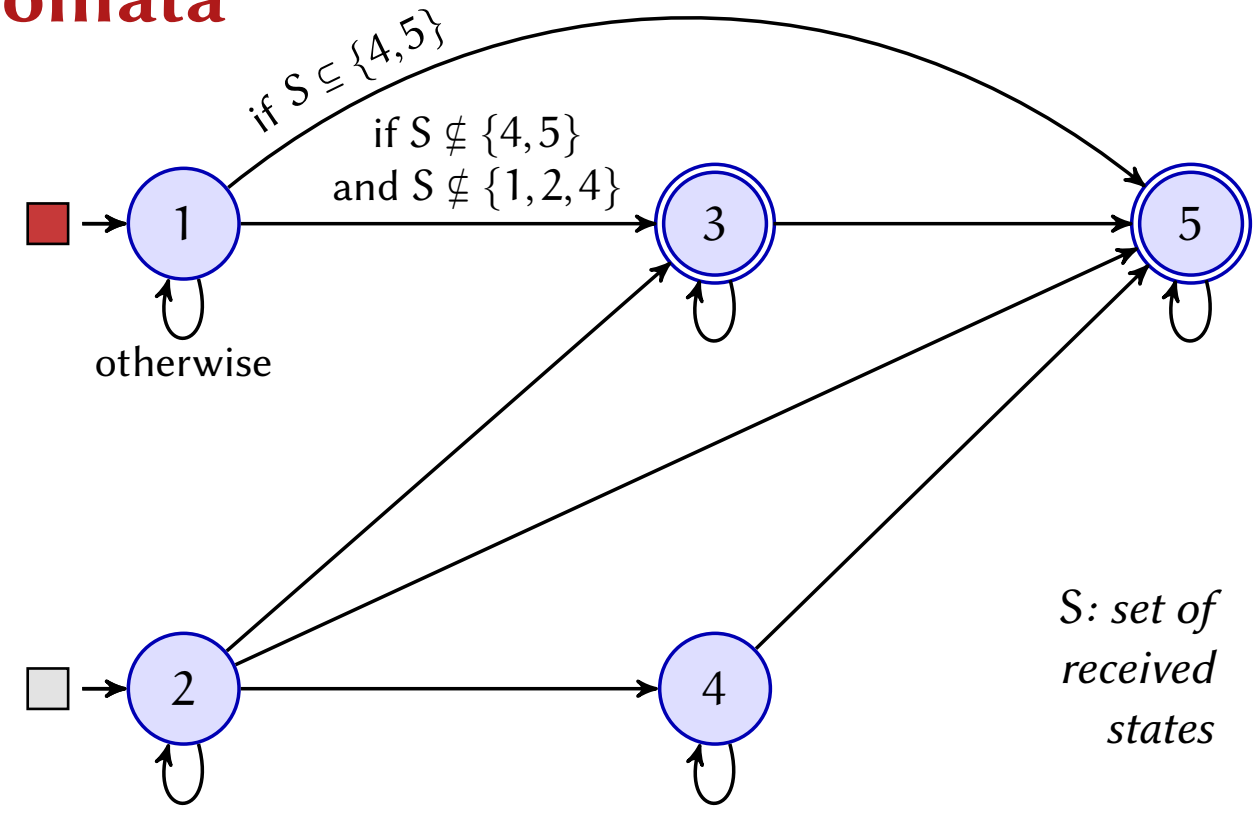


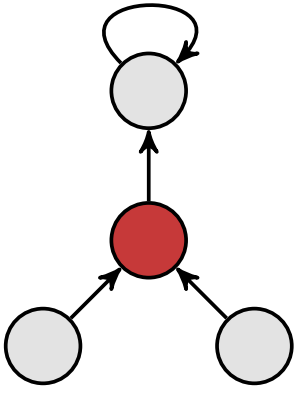


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

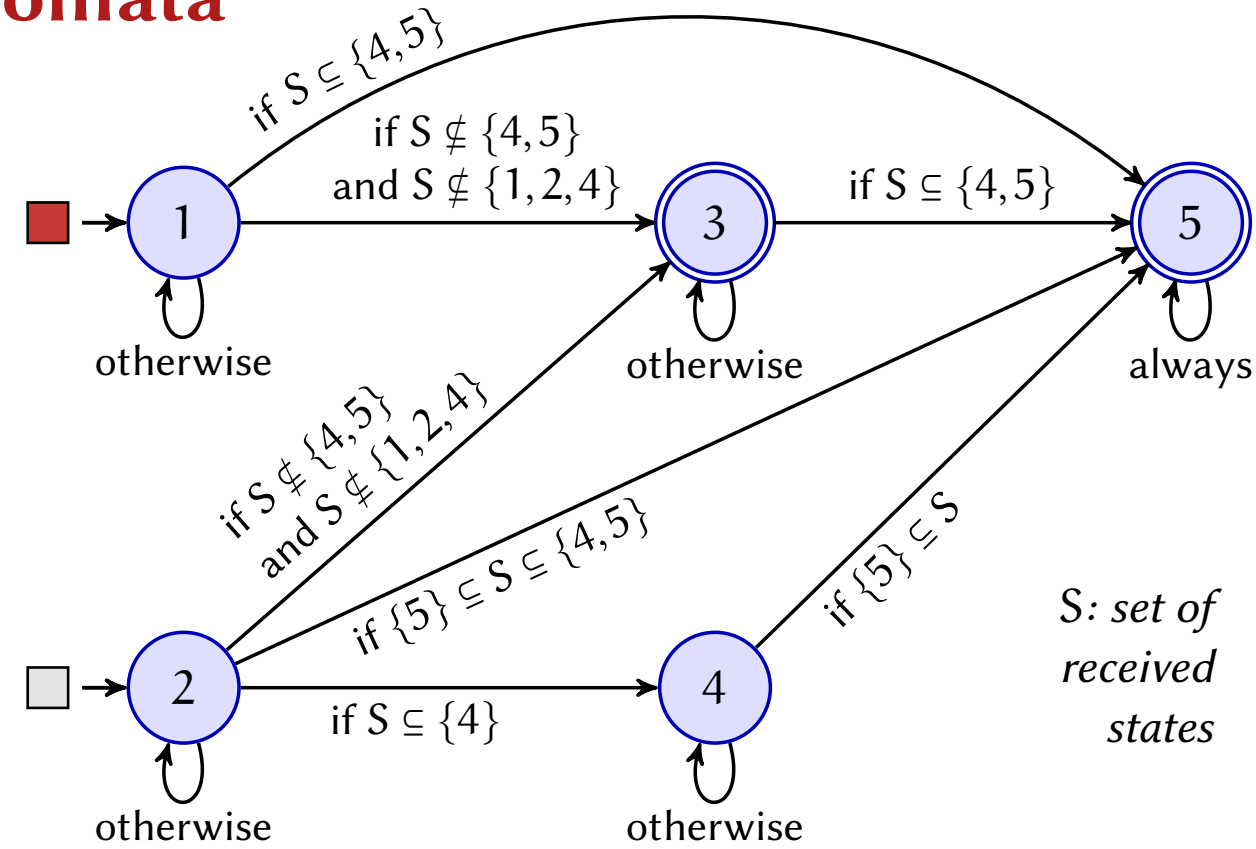


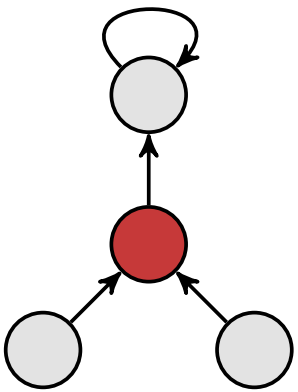


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

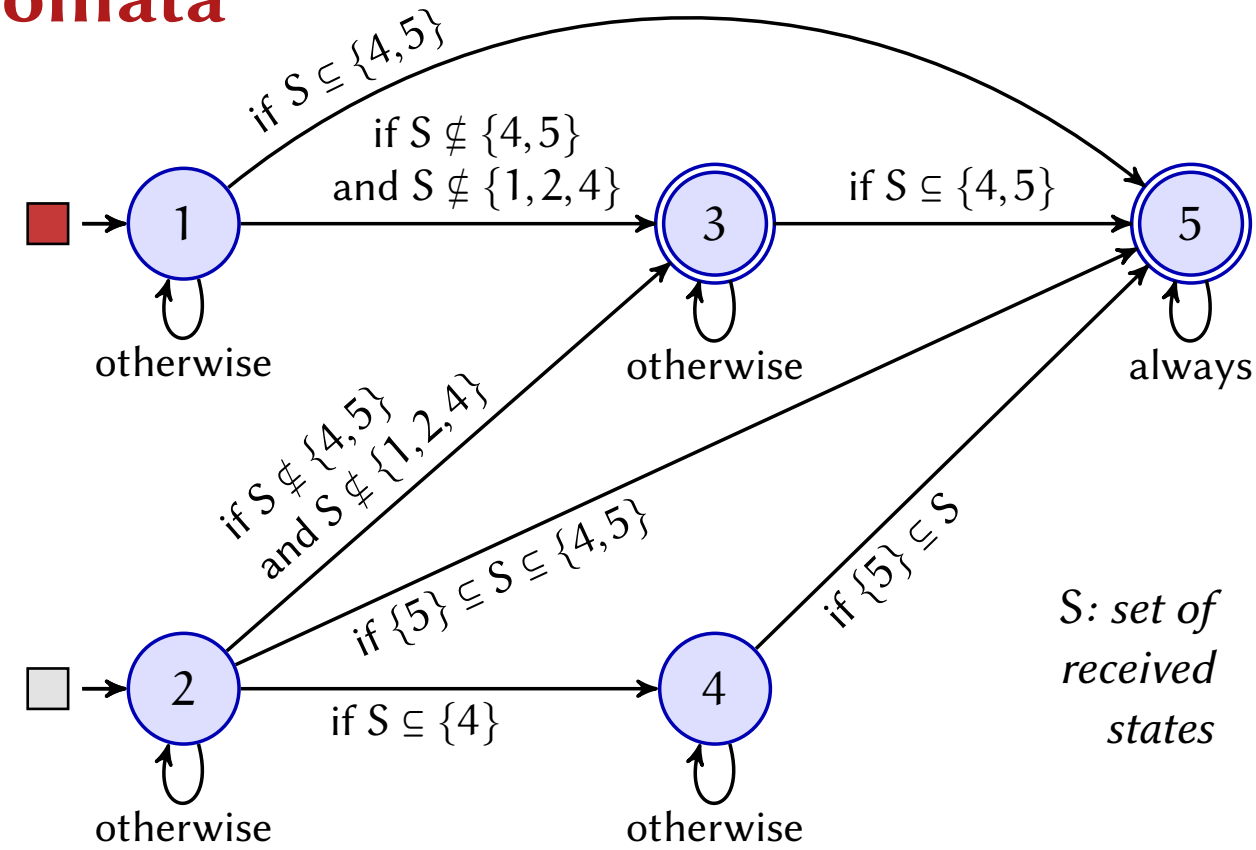


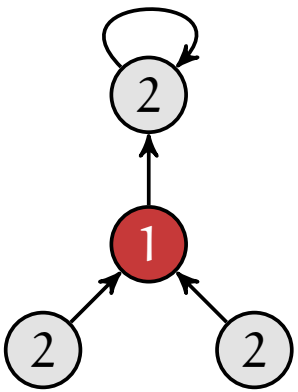


Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.



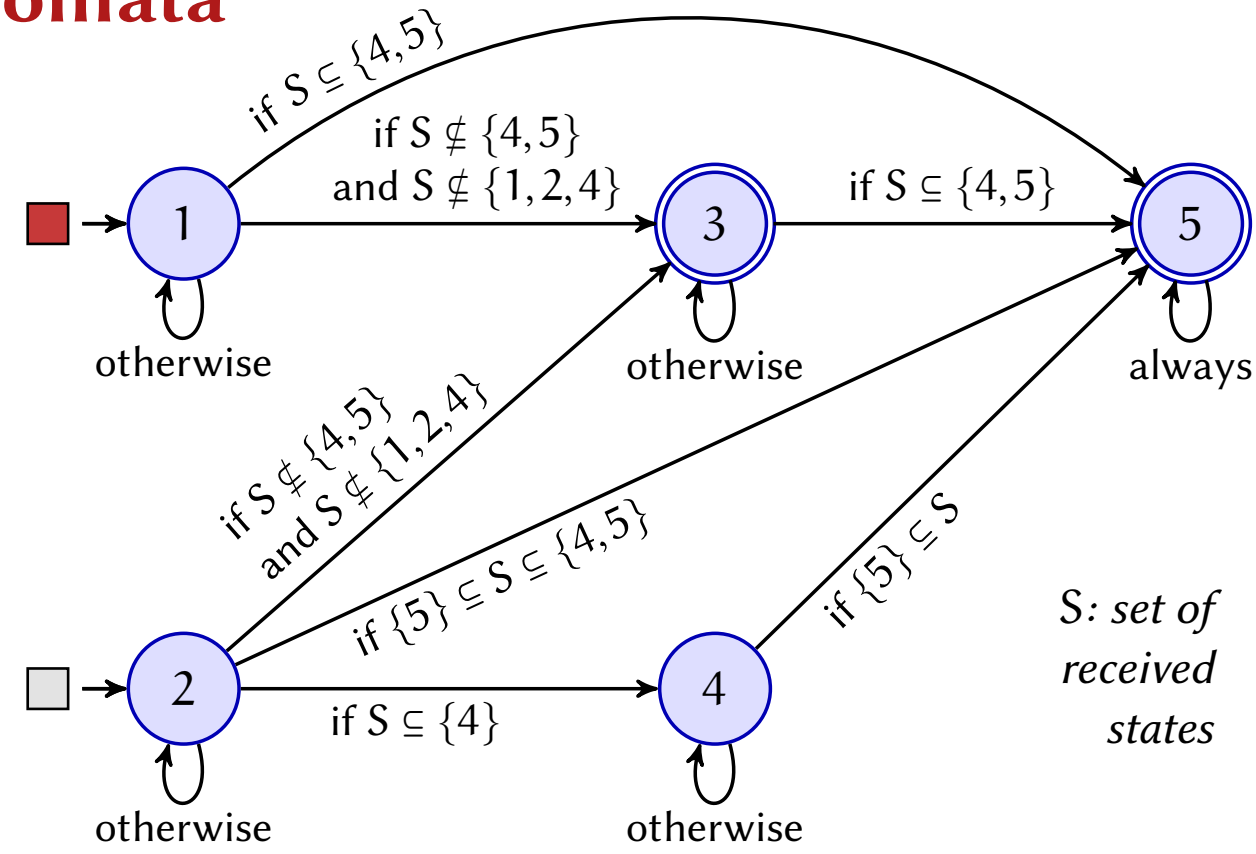


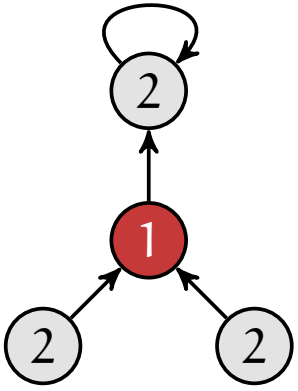
Asynchronous automata

$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.





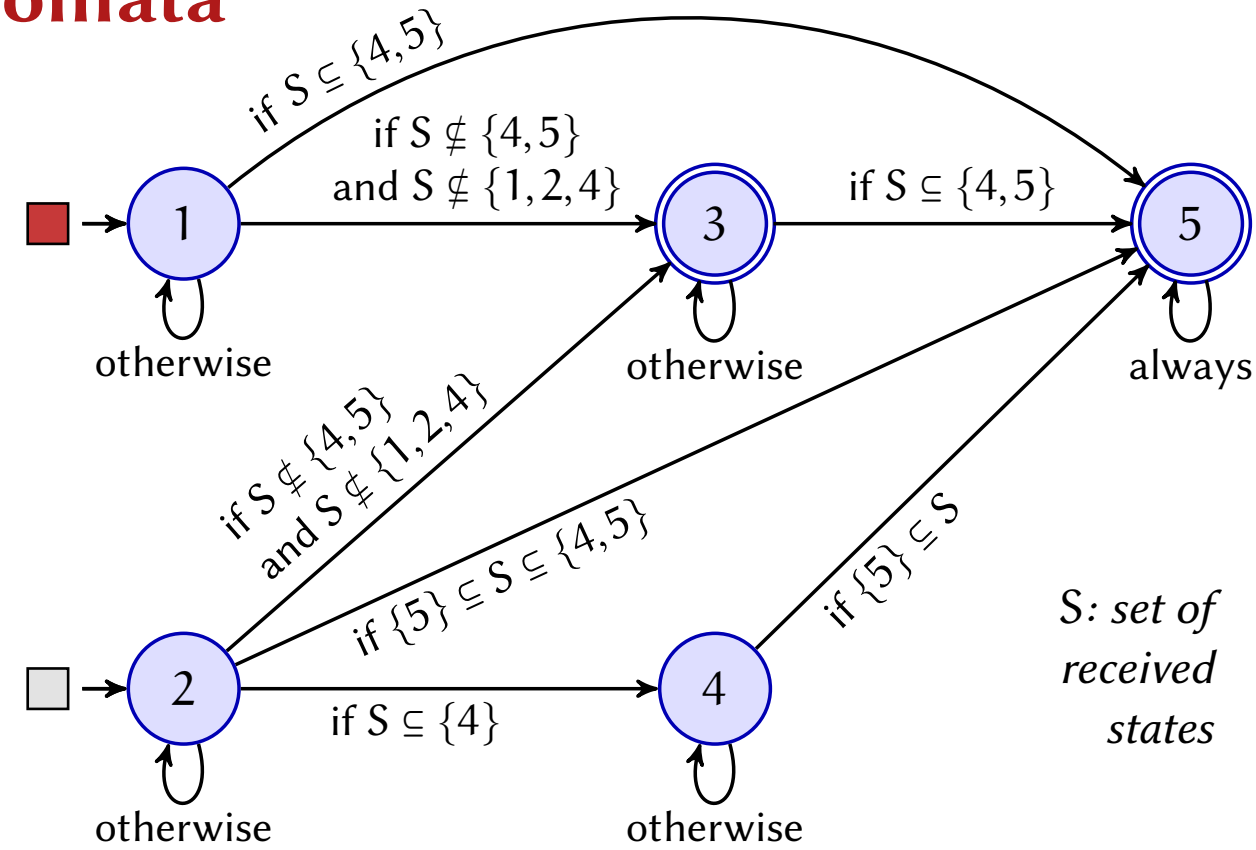
Asynchronous automata

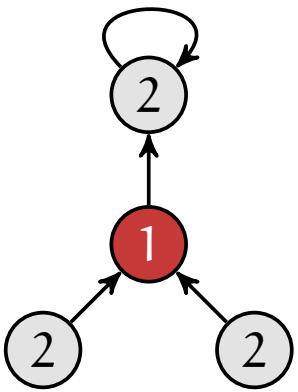
$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.

Messages may be delayed (FIFO).





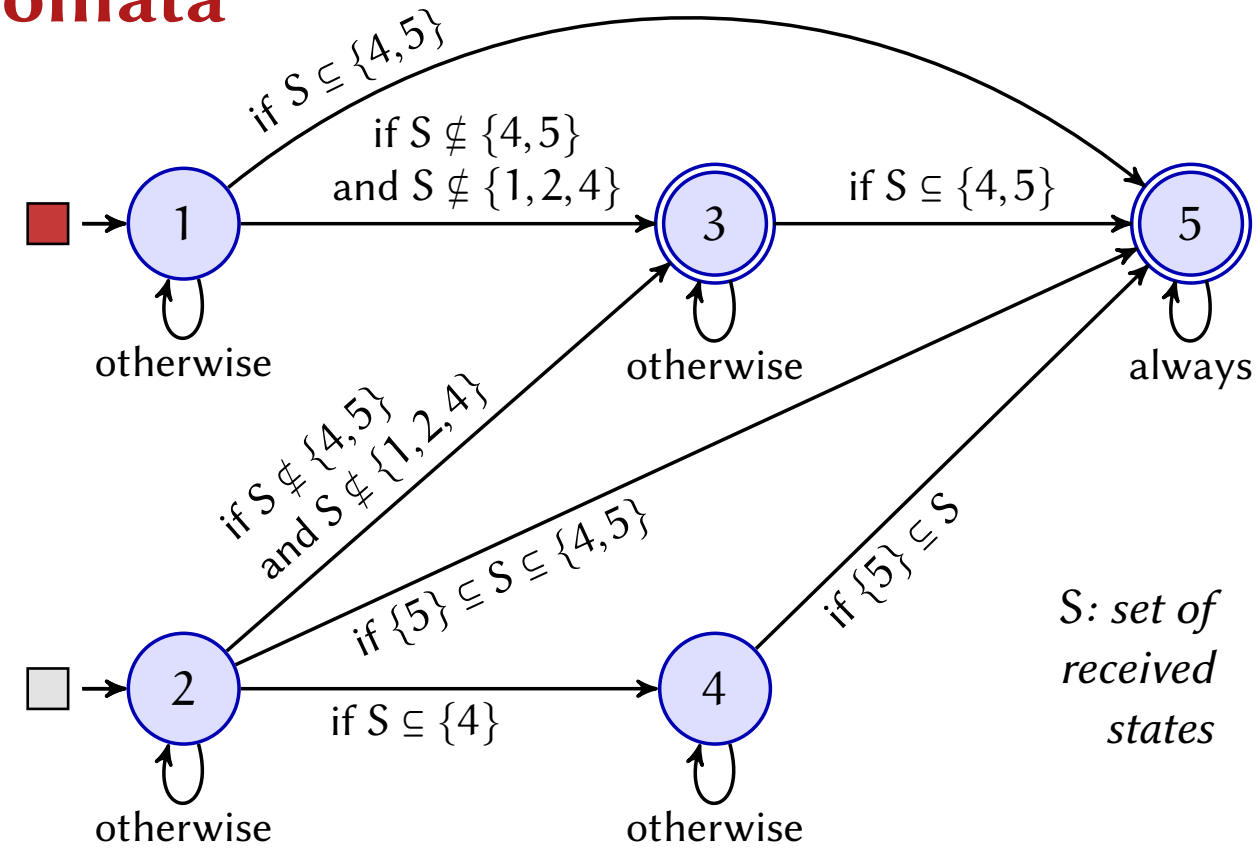
Asynchronous automata

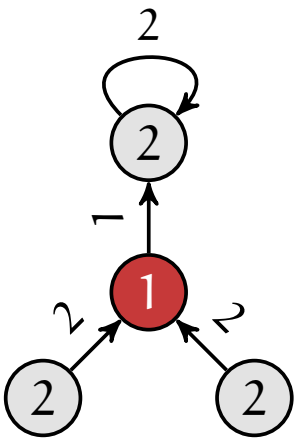
$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.

Messages may be delayed (FIFO).





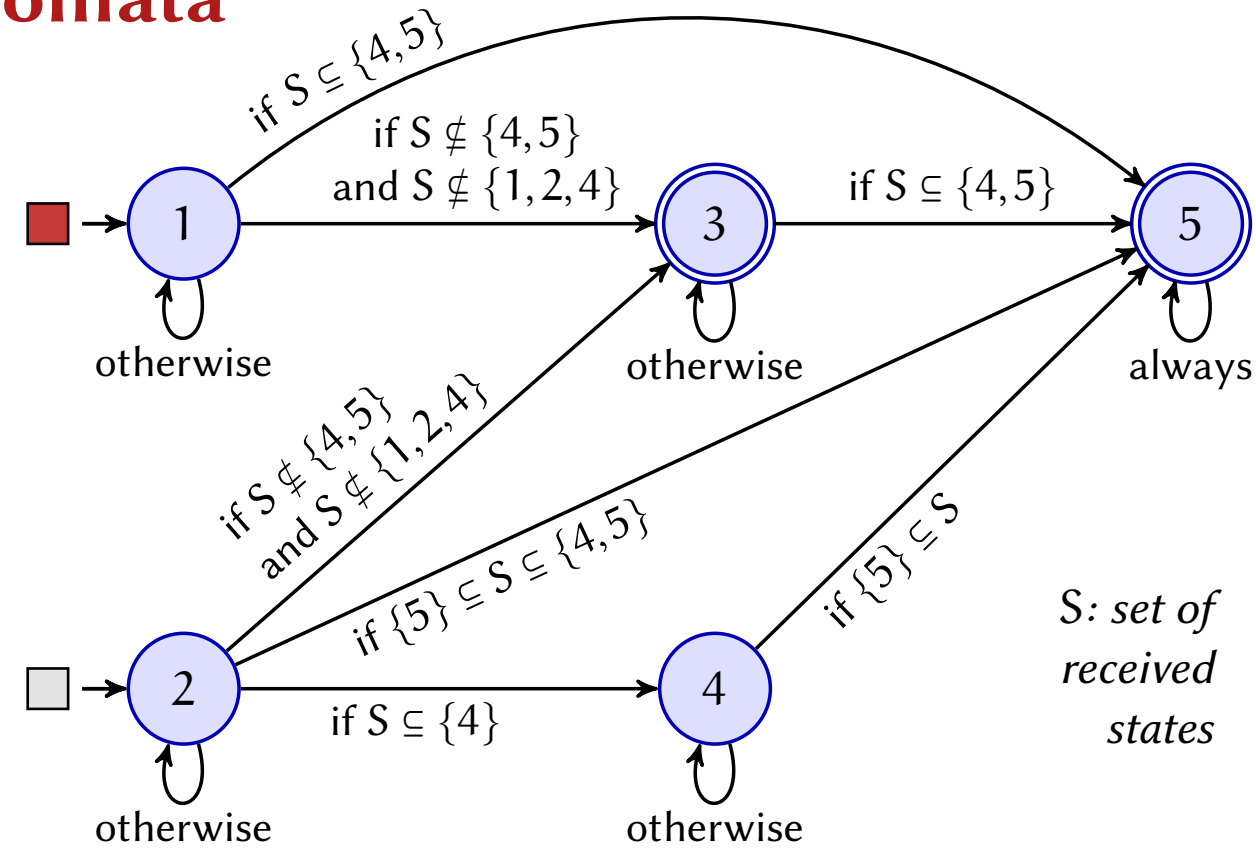
Asynchronous automata

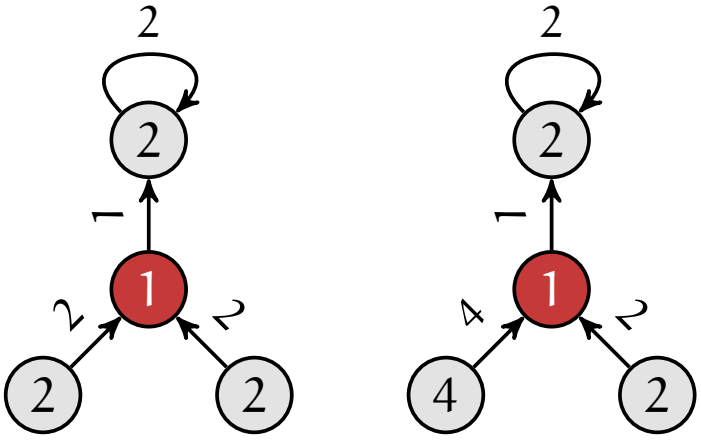
$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.

Messages may be delayed (FIFO).





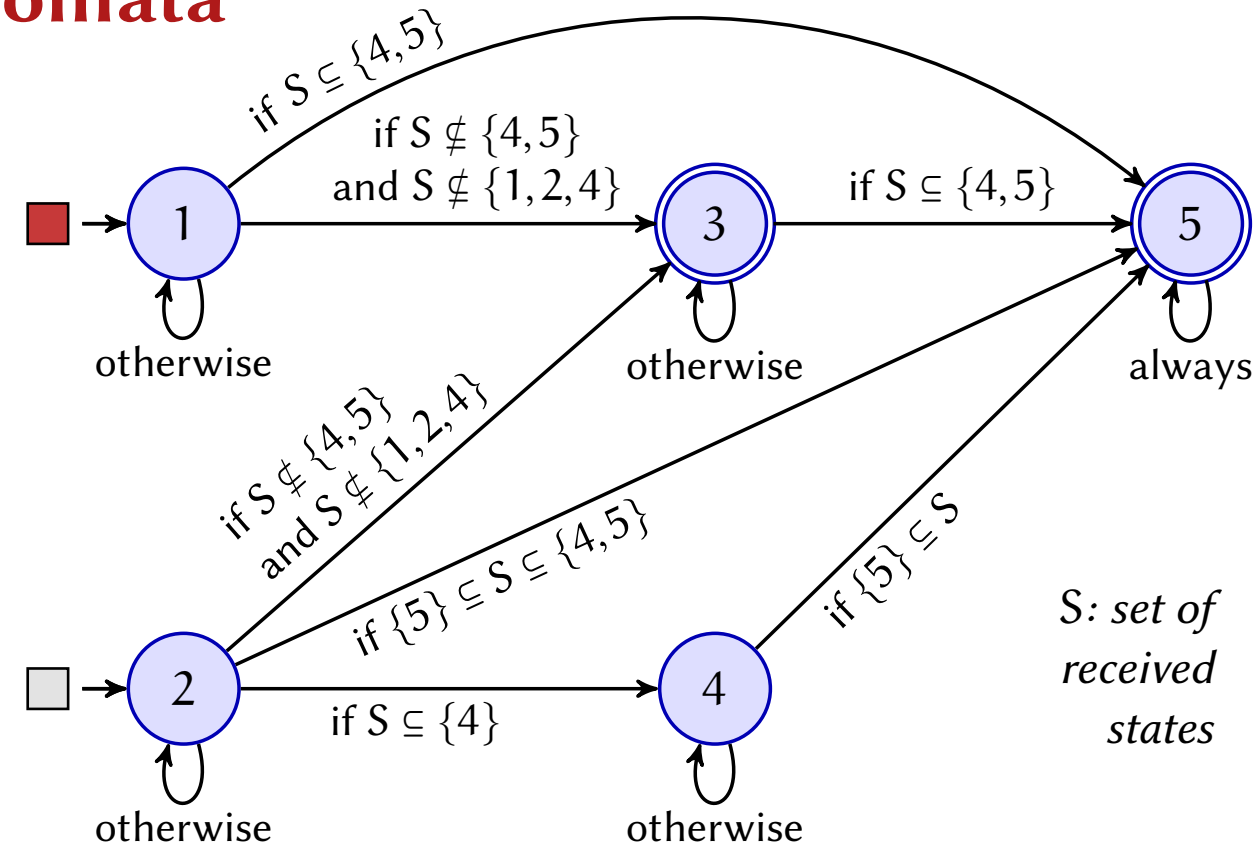
Asynchronous automata

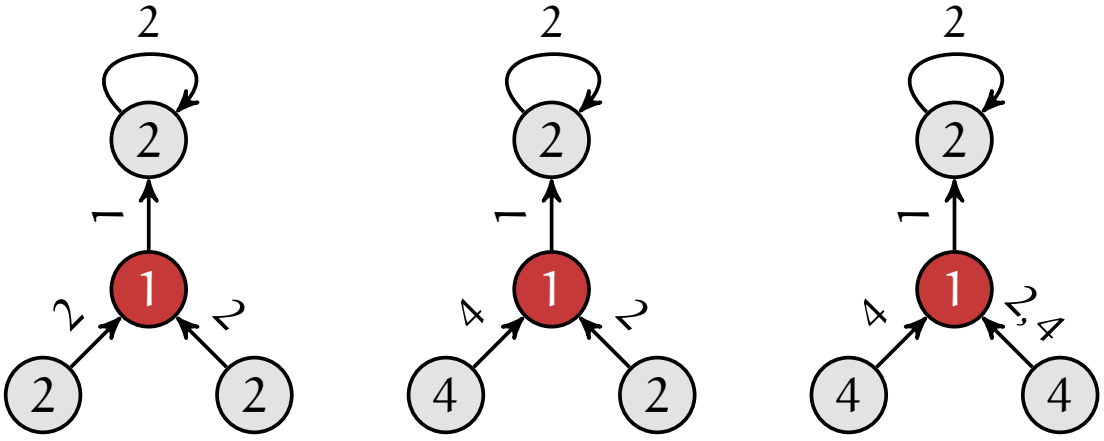
$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.

Messages may be delayed (FIFO).





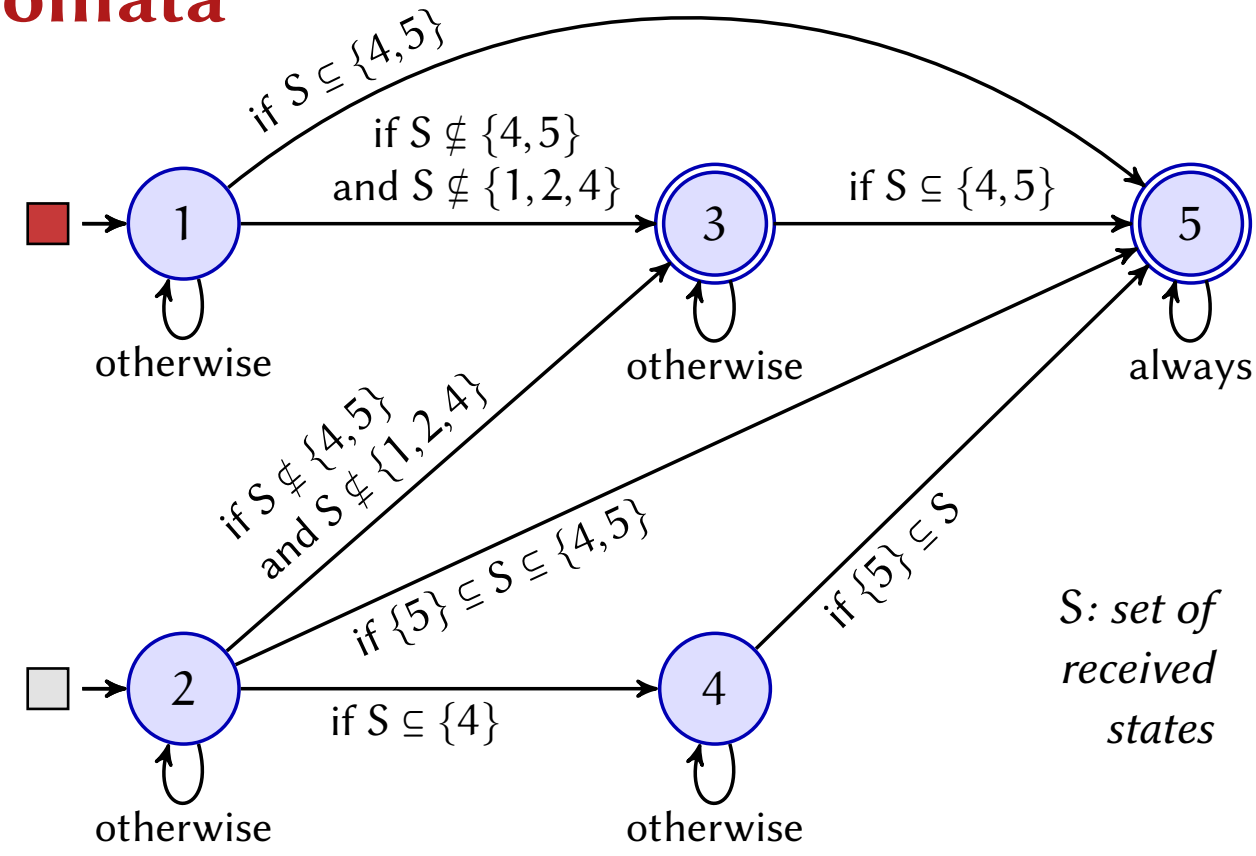
Asynchronous automata

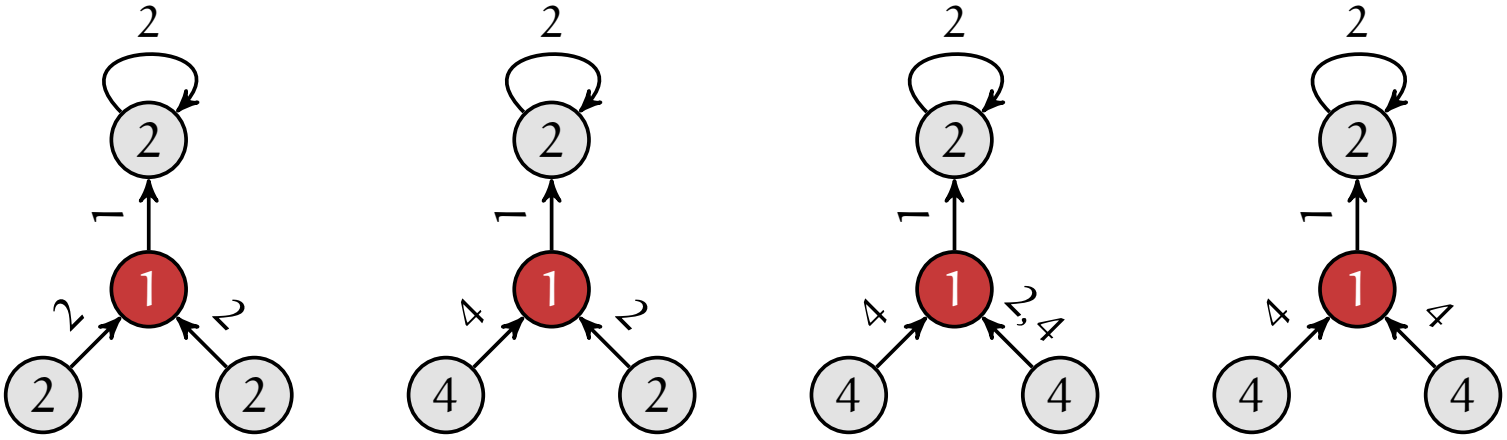
$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.

Messages may be delayed (FIFO).





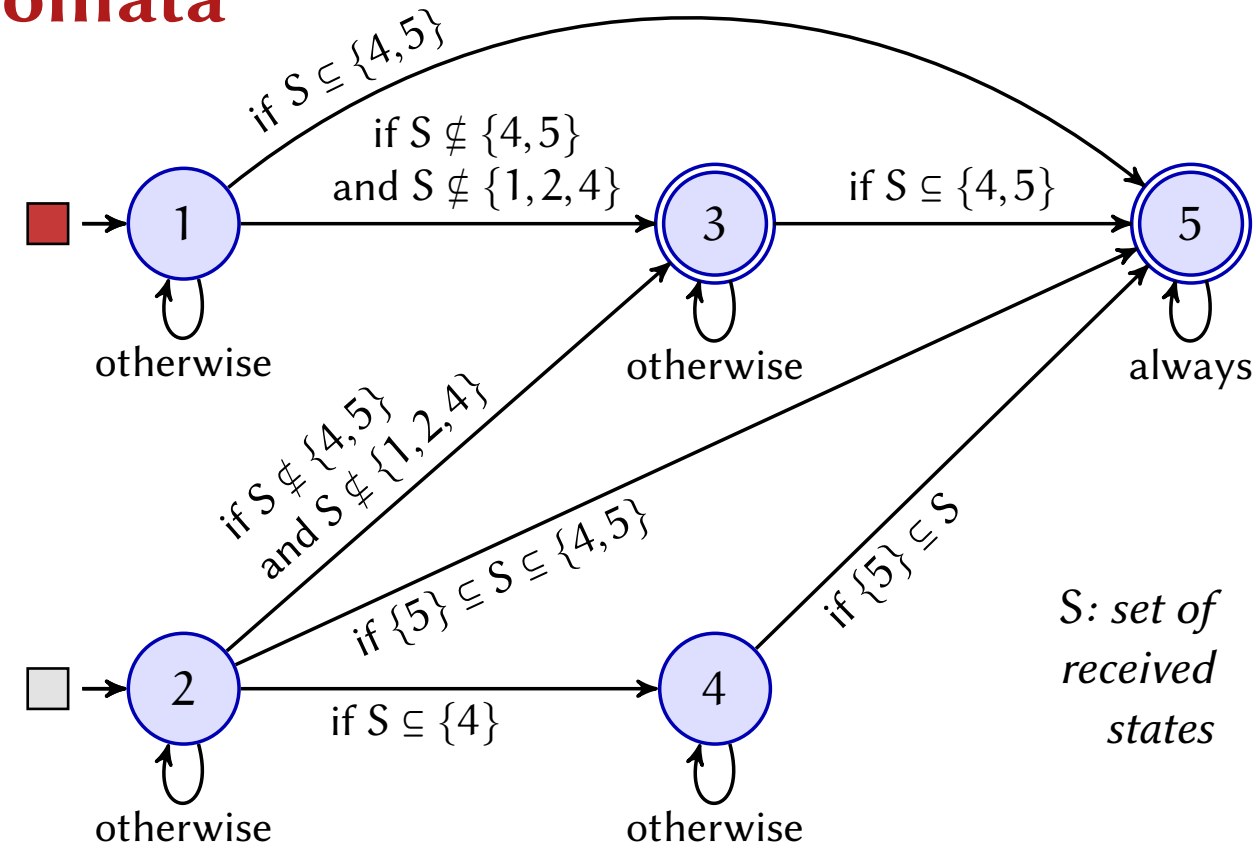
Asynchronous automata

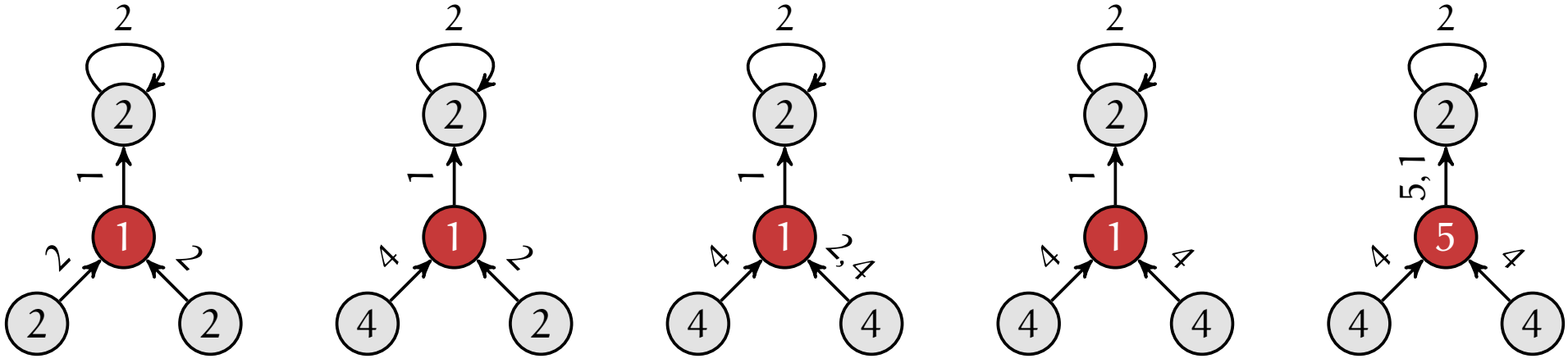
$\delta: Q \times 2^Q \to Q$

Quasi-acyclic diagram.

Nodes may sleep & miss messages.

Messages may be delayed (FIFO).





Definition of asynchrony

Definition of asynchrony

Definition of asynchrony

A malicious adversary can choose the timing, subject to fairness constraints.

Definition of asynchrony

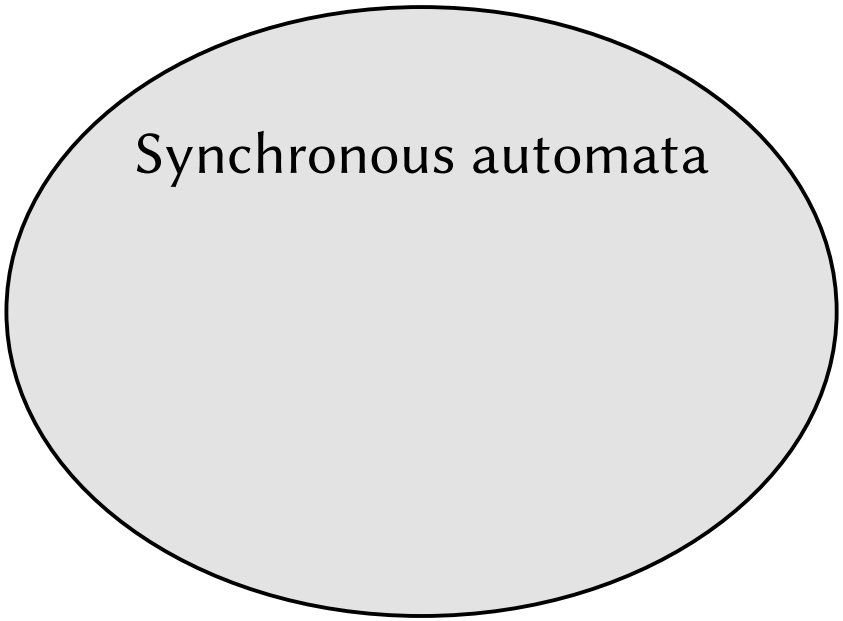
A malicious adversary can choose the timing, subject to fairness constraints.

An automaton is asynchronous if its acceptance behavior is independent of the adversary (on all finite digraphs).

Definition of asynchrony

A malicious adversary can choose the timing, subject to fairness constraints.

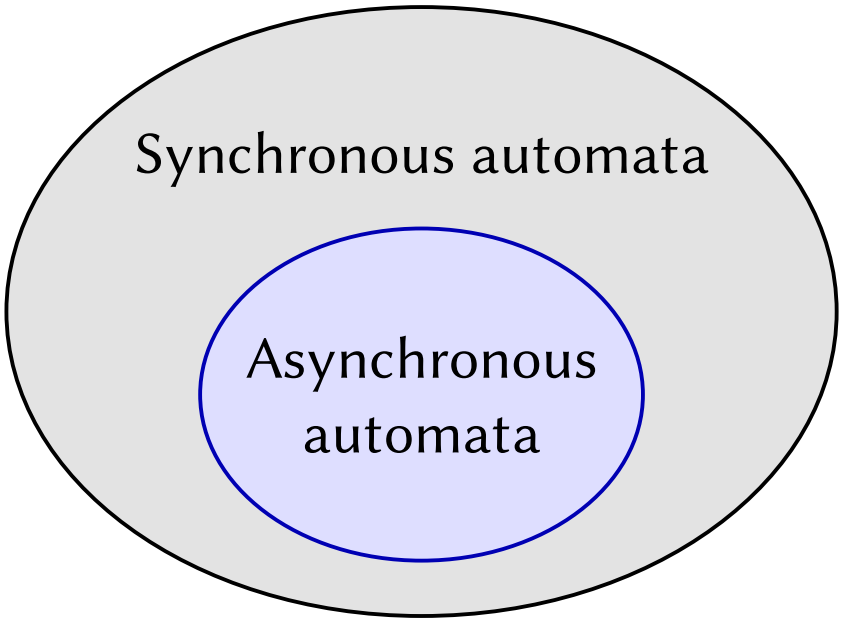
An automaton is asynchronous if its acceptance behavior is independent of the adversary (on all finite digraphs).



Definition of asynchrony

A malicious adversary can choose the timing, subject to fairness constraints.

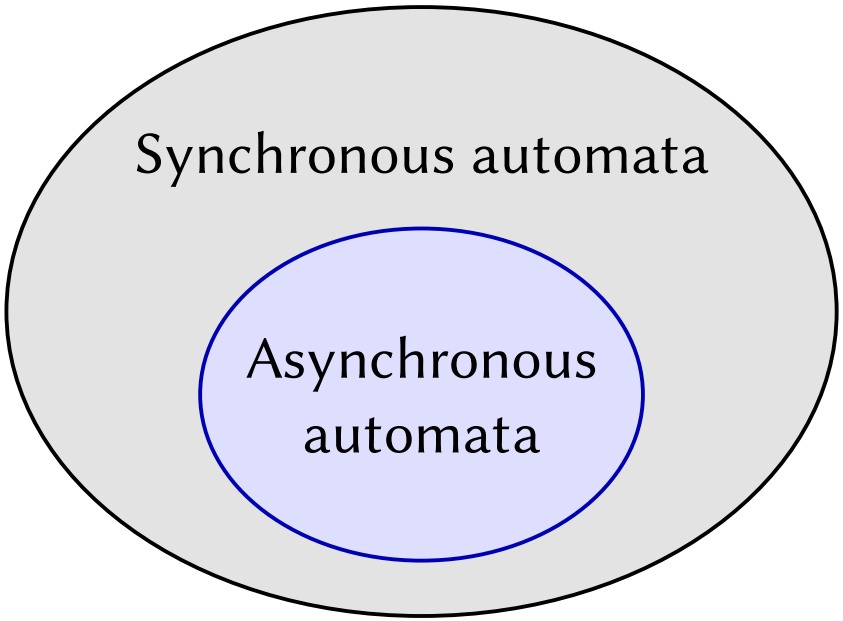
An automaton is asynchronous if its acceptance behavior is independent of the adversary (on all finite digraphs).



Definition of asynchrony

A malicious adversary can choose the timing, subject to fairness constraints.

An automaton is asynchronous if its acceptance behavior is independent of the adversary (on all finite digraphs).



Asynchrony is an additional semantic property.

Perspectives

Perspectives

- An alternation level that covers first-order logic?

Perspectives

- An alternation level that covers first-order logic?
- Can we decide if an automaton is asynchronous?

Perspectives

- An alternation level that covers first-order logic?
- Can we decide if an automaton is asynchronous?
- ...

Perspectives

- An alternation level that covers first-order logic?
- Can we decide if an automaton is asynchronous?
- …

- A “Fagin-style” theorem for distributed computing?

Perspectives

- An alternation level that covers first-order logic?
- Can we decide if an automaton is asynchronous?
- …

- A “Fagin-style” theorem for distributed computing?

2019…

Perspectives

- An alternation level that covers first-order logic?
- Can we decide if an automaton is asynchronous?
- …

- A “Fagin-style” theorem for distributed computing?

2019…

Thanks!